Beyond Heavy Cryptography: Practical Private Vector Search

Chen Gong, Xiangyu Wang, *Member*, *IEEE*, Dan Zhu, *Member*, *IEEE*, Yuanbo Guo, *Member*, *IEEE*, Chunjie Cao, *Member*, *IEEE*, Jianfeng Ma, *Member*, *IEEE*

***Abstract*—Driven by the explosion of AI applications, private vector search has recently gained widespread attention, but it still faces performance bottlenecks. Graph-based private search systems (Zhu et al., OSDI 2025) suffer from multi-round communication latency, while IVF-based systems (Henzinger et al., SOSP 2023) incur recall degradation and heavy computational overhead. In this paper, we propose EHSearch, a single-round, high-throughput private vector search system on encrypted data. EHSearch introduces HGI, an index structure that distills the HNSW graph into fixed-size groups, preserving local navigability for high recall. To avoid heavy cryptographic primitives, EHSearch presents Compute-Free Encrypted Filtering, which couples binary quantization with block-scoped pigeonhole matching, reducing the server workload from dense encrypted arithmetic to lightweight exact lookups. Extensive evaluations across four real-world datasets demonstrate that EHSearch outperforms state-of-the-art private baselines by 2 to 5 orders of magnitude in throughput, and scales out-of-core to 1 billion vectors on a single SSD. Specifically, when achieving 90% Recall@10 on 1 million points, EHSearch answers a private vector search in 22.35ms P99 latency in LAN, and sustains 223.3ms P99 latency in WAN environments.**

***Index Terms*—Approximate nearest neighbor search, private vector search, encrypted data.**

I. Introduction

Approximate nearest neighbor search (ANNS) has emerged as a cornerstone of modern artificial intelligence (AI) applications. Diverse multimodal data are increasingly represented as high-dimensional embeddings, in which semantic proximity is encoded as vector closeness [1], [2]. Consequently, efficiently searching these embedding vector spaces via ANNS is critical for enabling AI retrieval and reasoning. To address the computational demands of searching over massive embeddings, specialized cloud-native vector databases have emerged as critical infrastructure [3], [4], [5]. However, outsourcing sensitive embeddings to third-party cloud platforms introduces critical privacy risks. Mounting evidence suggests that embeddings are vulnerable to inversion attacks, allowing adversaries to recover raw data [6], [7], [8]. To address these privacy issues, *private vector search*, which protects embeddings against inversion attacks, has recently attracted widespread attention, yet remains costly [9], [10], [11].

Existing private vector search systems generally follow a common model: the server hosts the encrypted vectors, while the client maintains cryptographic keys and routing metadata. As shown in Figure 1, the client's ANNS procedure relies on accessing server-side data on demand, requiring the server to perform *secure filtering* to identify relevant encrypted vectors without learning their contents. The secure filtering step is the central performance bottleneck, as its efficiency is fundamentally determined by the underlying index structure. Current systems adopt one of two index families: graphs or inverted files (e.g., IVF [12], [13]), but both incur considerable costs for secure filtering.

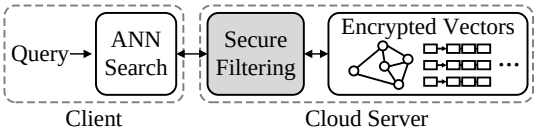


Fig. 1: The common system model.

Graph-based systems [11], [10] preserve the graph topology to maintain high recall. However, securely traversing a graph inherently dictates iterative, data-dependent memory accesses. This sequential dependency forces the client to execute multiple sequential network round-trip times (RTTs) per query, incurring considerable latency. To conceal traversal paths from the server, state-of-the-art schemes deploy heavy cryptographic primitives, such as Oblivious RAM (ORAM) [14], [15], [11] or Private Information Retrieval (PIR) [16], [17], [10]. These primitives impose huge overheads on communication volume and computation, rendering them impractical for large-scale serving.

IVF-based systems [9], [18] avoid multi-round interactions by grouping vectors into flat clusters via K-Means, enabling the client to fetch all target clusters in a single batch. However, this approach suffers from two severe setbacks. First, geometric K-Means partitioning severs long-range cross-region edges essential to graph navigability: a query's true nearest neighbors frequently span cluster boundaries and are truncated, resulting in sharp recall degradation. To recover recall, the client must retrieve an inflated number of clusters, which substantially increases network bandwidth usage. Second, to conceal cluster selection, existing single-round schemes resort to Homomorphic Encryption (HE) [19], [20]. Evaluating dense ciphertext-matrix multiplications over HE creates an immense computational bottleneck.

Chen Gong, Xiangyu Wang, and Jianfeng Ma are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China (e-mail: cgong_11@stu.xidian.edu.cn, xywang_xidian@163.com, jfma@mail.xidian.edu.cn).

Dan Zhu is with the School of Cyber Security, Northwestern Polytechnical University, Xi'an 710071, China (e-mail: zhudan@nwpu.edu.cn).

Yuanbo Guo and Chunjie Cao are with the School of Cyber Security, Hainan University, Haikou 570228, China (e-mail: yuanbog@hainanu.edu.cn, caochunjie@hainanu.edu.cn).

Corresponding author: Xiangyu Wang.

Overall, existing systems fall into two fundamental misalignments. *(1) Structural mismatch:* Directly adopting plaintext index structures into encrypted environments without co-designing the index layout and the filtering mechanism. *(2) Cryptographic over-design:* Consistently pursuing near-zero leakage by deploying heavy cryptographic primitives, treating high-dimensional embeddings under the same threat assumptions as low-dimensional structured data. This prompts a fundamental question:

Is it possible to build a cost-efficient private vector search system that achieves single-round retrieval and high recall?

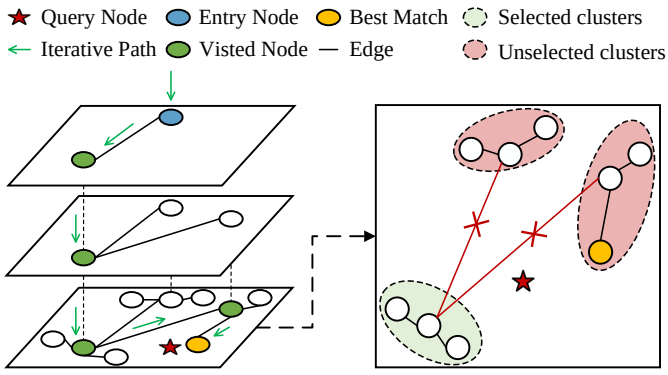
In this paper, we present **EHSearch**, a high-throughput, single-round private vector search system that addresses this question. Our main contributions are as follows:

- **Topology-Distilled Data Layout:** We introduce HGI, an index structure that packs the HNSW graph into fixed-size, overlapping blocks. This approach preserves graph navigability and avoids boundary truncation, achieving high recall within a 1-RTT network. Crucially, we verify the structural correctness and completeness of HGI. HGI achieves 100.00% Layer-0 node reachability across all parameter settings, leaving no unreachable orphan vectors (§IV).
- **Compute-Free Encrypted Filtering:** Bypassing heavy cryptography, we couple randomized binary quantization with block-scoped multi-indexes to convert distance filtering into $O(1)$ exact lookups. The mechanism provably guarantees adaptive security under standard symmetric cryptography (§V).
- **Implementation and Evaluation:** We realize EHSearch as a self-contained C++ system that supports both high-performance in-memory serving and I/O-optimized out-of-core SSD retrieval. To evaluate real-world practicality, we benchmark EHSearch across diverse semantic workloads under both LAN and WAN environments. Evaluations show that EHSearch outperforms state-of-the-art private baselines by 2 to 5 orders of magnitude in throughput and sustains stable millisecond-level tail latency (P95/P99). EHSearch also preserves sub-second responsiveness over WAN via its strict 1-RTT design and scales out of core to 1 billion vectors on an SSD with bounded physical I/O (§IX).

II. Background and Motivation

A. HNSW and the Challenge of Iterative Search

The Hierarchical Navigable Small World (HNSW) [21] graph is widely used for ANNS due to its high accuracy and efficiency. HNSW organizes objects into a multi-layer proximity graph. The sparse upper layers act as a global routing mechanism, while the dense bottom layer (Layer 0) ensures accurate local search. During construction, edges are established using heuristics similar to the Relative Neighborhood Graph [22], [23], which connects proximate nodes while deliberately preserving long-range edges to maintain small-world navigability. During a query, the search employs greedy routing: it calculates distances from the query to the current node's neighbors and transitions to the closest neighbor. This process repeats iteratively until a local minimum is reached in the bottom layer.



(a) HNSW (b) Directly Cluster Layer 0

Fig. 2: Structural bottlenecks of existing schemes in encrypted environments. (a) Data-dependent routing requires multiple sequential network round-trip. (b) Direct clustering on Layer 0 removes essential long-range edges, thereby isolating the true nearest neighbors and reducing recall.

The Challenge of Iterative Graph Search. While HNSW achieves logarithmic time complexity in local memory, its search procedure is inherently a *data-dependent sequential process*. The selection of the $(i+1)$-th node depends strictly on the distance computations of the i-th node's neighbors. As shown in the left part of Figure 2, deploying HNSW on encrypted data [11], [10] introduces two structural bottlenecks:

1) *Multi-Round Interactions:* In a private search system, the client cannot pre-fetch all required nodes. A graph traversal of L hops requires $O(L)$ sequential network RTTs or cryptographic operations. In WAN, accumulated RTT strictly limits search latency.
2) *Client Storage Overhead:* The graph topology reveals semantic relationships among vector points. To prevent the server from inferring these relationships via traversal paths, existing private search schemes usually require the client to store the graph connectivity locally. For billion-scale datasets, caching this structural metadata incurs significant memory overhead.

Insight 1. *The fundamental bottleneck of graph-based encrypted search lies in its data-dependent sequentiality.* Even when systems like Compass [11] cache the sparse upper layers on the client, searching within the massive Layer 0 (which hosts $> 97\%$ of all vectors) still requires iterative, hop-by-hop traversals over ORAM, incurring $O(L)$ sequential network round trips. However, we observe a key property: while a query's traversal path across Layer 0 is temporally dynamic, the local topology surrounding each upper-layer entry point is predetermined and spatially static. During HNSW construction, Layer-0 nodes naturally cluster around their incoming anchors. Therefore, instead of iteratively fetching nodes along an unpredictable query path over multiple RTTs, we can pre-pack these static topological neighborhoods into fixed-size spatial blocks. This transforms unpredictable multi-hop

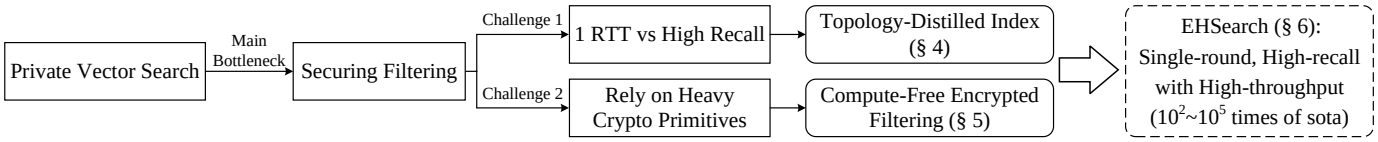


Fig. 3: Design overview of EHSearch.

routing into a deterministic, single-round spatial retrieval: the client identifies target entry points locally and fetches their pre-packed neighborhoods in a single batch, eliminating both multi-round network delays and client-side graph storage.

B. IVF-based Indexes and the Recall Decline

To avoid the iterative traversal bottlenecks, recent single-round private search systems [9], [18] adopt flat Inverted File (IVF) indexes. By applying standard clustering algorithms (K-Means), these systems group vectors into discrete partitions, allowing the client to fetch target clusters in a single batch, but at the cost of bandwidth waste and reduced recall.

Why Not Directly Cluster the Layer 0 of HNSW? As shown in the right part of Figure 2, directly applying hard partitioning introduces a conflict between geometric boundaries and topological navigability. First, K-Means strictly partitions space based on specific metrics, thereby severing the long-range edges that graph algorithms rely on to escape local optima. Second, a query's true nearest neighbors frequently span across cluster boundaries. When cross-cluster bridging edges are removed, nodes in unselected partitions are ignored entirely.

To recover from the resulting recall decline, IVF-based systems are forced to retrieve more clusters. This increase in retrieval volume raises network bandwidth requirements and the resulting ciphertext computation overhead, offsetting the advantages of single-round access.

C. Binary Quantization and Compute-Free Hamming Lookup

To mitigate the memory footprint and computational overhead of high-dimensional vectors, retrieval systems [3], [24] increasingly adopt vector quantization. RaBitQ [25] is a randomized binary quantization method that compresses D-dimensional float32 embeddings into compact D-bit binary codes, achieving a $32\times$ compression ratio.

RaBitQ normalizes input vectors to a unit hypersphere, applies a randomized orthogonal transformation, and maps the rotated vectors to the nearest vertices of a hypercube. This deterministic reduction ensures a monotonic correlation between other complex metrics (e.g., Euclidean similarity) and concise Hamming distance. Consequently, filtering vectors based on Euclidean proximity can be converted into searching for vectors within a bounded Hamming distance. However, computing Hamming distances directly over encrypted data (e.g., Homomorphic Encryption) still incurs significant server-side computational overhead in large-scale databases

Insight 2. To break this computational barrier, we shift the paradigm from encrypted distances computation to *encrypted lookup* for vectors that satisfy the distance constraint, which only requires deterministic encryption. We observe that if a target vector is within a Hamming distance of r from the query, there are at most r bit flips between their binary codes. By dividing a D-bit binary code into m disjoint sub-segments, the Pigeonhole Principle dictates that if $r < m$, at least one sub-segment must match perfectly without any bit flips. This property bridges the gap between approximate similarity and exact lookup: a Hamming distance constraint can be naturally decomposed into an exact-match condition on at least one sub-segment. Therefore, we can construct a multi-inverted index in which each subsegment serves as a key mapping to the vector IDs that contain it. During a query, the server simply uses the encrypted query sub-segments (trapdoors in § V-C) to perform $O(1)$ exact lookups in these inverted indexes. Any matched vector is guaranteed to fall within the Hamming radius r, fundamentally eliminating the need for distance calculations over encrypted data.

III. Design Overview

Figure 3 sketches the design of EHSearch. By integrating the Topology-Distilled Index (HGI) with co-designed Compute-Free Encrypted Filtering, we propose **EHSearch**, a cost-efficient private vector search system.

As previously analyzed, the performance bottleneck in secure filtering for private vector search is the inability to efficiently reduce the search space without incurring multiple RTT delays or recall degradation. To address this challenge, we first design HGI, a vector index that achieves high recall with low overhead in a single communication round. HGI restructures the bottom layer (Layer 0) of an existing HNSW graph into fixed-size groups. By executing a Breadth-First Search (BFS) traversal originating from all fixed entry points of Layer 0, HGI distills the original graph topology into discrete partitions with natural, slightly overlapping boundaries. These groups are then outsourced to the server for secure storage, while the remaining sparse upper layers of the HNSW graph are preserved on the client. During a search, the client performs high-speed local routing to identify the target groups on the server. The client can then retrieve all target groups in a single batch, decrypting and re-ranking them locally to obtain the final results.

To further reduce bandwidth and computational overhead, we design a lightweight, encrypted filtering mechanism. This mechanism allows the server to perform secure filtering within the targeted groups, completely decoupled from the graph traversal state. Unlike the iterative, routing-dependent hopping in graph-based schemes, this filtering relies on O(1) exact lookups in the sub-segment indexes.

EHSearch's APIs are as follows:

- $\mathsf{Init}(\mathsf{DB})$: Given a raw vector database DB, initialize the index and data layout.
- $\mathsf{LocRout}(Q) \rightarrow$ Targets: Given a query Q, the client performs local routing to identify the target groups.
- $\mathsf{TrapGen}(Q, r) \rightarrow$ Trapdoors: Given a query Q and a radius r, the client generates encrypted trapdoors.
- $\mathsf{Filter}(\text{Trapdoors}, \text{Targets}) \rightarrow$ Candidates: Server uses trapdoors to perform encrypted lookup in target groups and returns encrypted candidate vectors.
- $\mathsf{Refine}(Q, \text{Candidates}) \rightarrow$ Result: The client decrypts the candidate vectors and performs exact refinement with Q to obtain the result.

A. Threat Model and Security Guarantees

We assume a fully trusted client and an honest-but-curious server, standard in private search [26], [27], [28], [11], [10], [9]. The server honestly executes the protocol but attempts to deduce raw embeddings or query semantics from stored ciphertexts and execution transcripts.

1) Security Definition: Let $\mathsf{DB} = \{v_1, \ldots, v_N\}$ be the database partitioned by HGI into t groups $\mathsf{Group}(\mathsf{DB}) = \{\mathcal{G}_1, \ldots, \mathcal{G}_t\}$, each padded to a uniform capacity B via overlapping boundary padding (§IV-C). For each group $\mathcal{G}_j$, a Group-Scoped Multi-Index $\mathsf{ST}_j = \{T_{j,1}, \ldots, T_{j,m}\}$ maps sub-segment pseudo-random function (PRF) [29] evaluations to PRF-masked vector IDs. Raw vector payloads are encrypted using AES-CBC with IND-CPA security. We parameterize observable search metadata via a formal leakage profile:

$$\mathcal{L}(\mathsf{DB}, \vec{q}) = \big(\mathcal{L}^{Setup}(\mathsf{DB}),\ \mathcal{L}^{Query}(\vec{q})\big), \tag{1}$$

where $\mathcal{L}^{Setup}(\mathsf{DB}) = |\mathsf{DB}|$ (the vector count), and $\mathcal{L}^{Query}(\vec{q}) = \big(\mathsf{AccessPattern}(\vec{q}),\ \mathsf{SearchPattern}(\vec{q})\big)$.

Definition 1 (Access Pattern and Search Pattern)**.** *For an adaptive sequence of queries $\vec{q} = (q_1, \ldots, q_n)$, we define:*

- $\mathsf{AccessPattern}(\vec{q})$*: The sequence of accessed group identifiers and returned candidate vector identifiers.*
- $\mathsf{SearchPattern}(\vec{q})$*: The equivalence relation indicating whether any two queries generate identical trapdoors.*

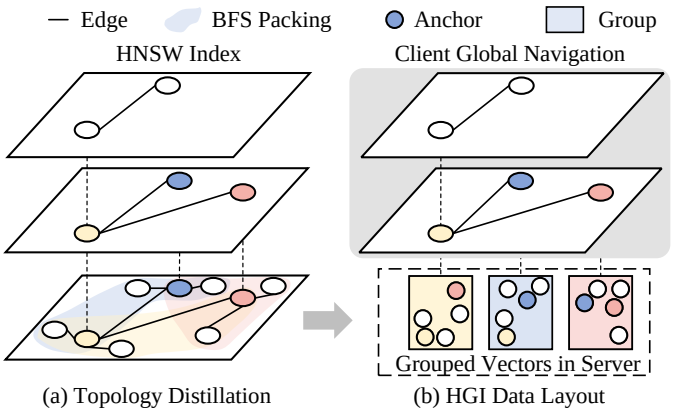
These patterns expose only abstract positional mappings; the underlying content is protected by AES-CBC's IND-CPA security and the PRF's pseudorandomness.

Definition 2 (Adaptive Security)**.** *For any PPT adversary $\mathcal{A}$ making a polynomial number of queries, we say a system $\Pi = (\mathsf{Init}, \mathsf{LocRout}, \mathsf{TrapGen}, \mathsf{Filter}, \mathsf{Refine})$ is $\mathcal{L}$-adaptively secure if there exists a PPT simulator $\mathcal{S}$ such that:*

$$\big|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]\big| \leq \mathsf{negl}(\lambda). \tag{2}$$

Here, $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ denotes $\mathcal{A}$'s view interacting with the real system on DB*, receiving target groups and PRF trapdoors for adaptively chosen queries q_i. $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ denotes the view synthesized by $\mathcal{S}$ given solely $\mathcal{L}$: $\mathcal{S}$ generates dummy groups from $|\mathsf{DB}|$ and simulates trapdoors and lookup responses matching $\mathsf{AccessPattern}(q_i)$ and $\mathsf{SearchPattern}(q_i)$.*

Theorem 1. *EHSearch is $\mathcal{L}$-adaptively secure if AES-CBC is IND-CPA secure and the PRF is cryptographically secure.*



(a) Topology Distillation (b) HGI Data Layout

Fig. 4: Data layout transformation via Topology Distillation. (a) HGI extracts upper-layer nodes as routing anchors and performs BFS packing on Layer 0. HGI encapsulates long-range edges and creates natural overlapping boundaries. (b) The resulting fixed-size groups are outsourced to the server, while the sparse upper layers are preserved on the client to serve as a lightweight global navigation index.

A proof sketch is provided in §VII-B, with the full proof using a hybrid argument detailed in Appendix A.

Discussion on Leakage vs. Practical Security. EHSearch relies strictly on symmetric encryption primitives, thereby revealing no metric coordinates or relative rankings. Enforcing uniform group capacity B further neutralizes volume-matching attacks [30], [31]. Regarding pattern leakage, known state-of-the-art reconstruction attacks [32], [33], [34], [35] have demonstrated coordinate recovery only on low-dimensional spatial points ($\leq$ 4D). To our knowledge, no existing attack can reconstruct high-dimensional (100+D) dense embeddings from discrete, unranked identifier overlaps alone, where severe distance concentration and unconstrained rotational ambiguity create formidable practical barriers (further analyzed in Appendix A-C). While releasing access patterns does not provide oblivious zero-leakage, the lack of practical high-dimensional reconstruction attacks makes bounded leakage an acceptable trade-off to eliminate heavy cryptographic bottlenecks.

IV. Topology-Distilled Index Construction

To achieve single-round retrieval without sacrificing the navigability of graph-based indexes, the continuous and interconnected graph topology must be transformed into discrete, fixed-size blocks suitable for encrypted storage. In this section, we detail the construction of the HNSW Grouped Index (HGI). We present a data layout transformation paradigm, *Topology Distillation*, that captures the essential routing capabilities of the HNSW graph by partitioning it into independent partitions via entry-point anchoring (§ IV-A), topology-preserving traversal (§ IV-B), and overlapping boundary padding (§ IV-C).

A. Entry-Point Anchoring

Traditional IVF-based systems [12], [13], [9] typically utilize unsupervised clustering algorithms, such as K-Means, to define partition centroids. However, these artificial geometric

centroids are computed blindly, without accounting for the dataset's underlying connectivity, leading to suboptimal routing in small-world networks. Instead of generating artificial centroids, HGI utilizes the hierarchical routing structure of HNSW [21]. In the HNSW construction, the upper sparse layer (Layer ≥ 1) serves as a global navigation, guiding new nodes to the dense local neighborhood of the lower layer (Layer 0). The data points in Layer 0 naturally form a local clustering structure around the upper incoming nodes.

HGI extracts all nodes from Layer ≥ 1 to serve as the navigation for our discrete groups in Layer 0. Because the upper layers contain only a small fraction ($1\% \sim 3\%$) of the total dataset, this sparse hierarchical graph can be preserved entirely in plaintext on the client side with a minimal memory footprint. This enables the client to perform rapid local routing to identify the most relevant anchors in Layer 1 without initiating any network requests, while leaving the massive, dense Layer 0 to be partitioned and outsourced to the server.

B. Topology-Preserving BFS Packing

Once the anchors are identified, the next step is to group the nodes in Layer 0 into corresponding partitions. A naive approach would be to assign each node to its nearest anchor based on Euclidean distance (i.e., Voronoi partitioning [36], [37], [38]). However, this geometric hard partitioning severs the cross-region connections. In HNSW, long-range edges are deliberately constructed to allow the greedy search to escape local optima. Cutting these edges forces the search into a latency-recall trade-off.

To preserve the local navigability, HGI abandons Euclidean spatial partitioning in favor of Topology-Preserving Breadth-First Search (BFS) Packing. For each anchor node e_i, we initiate a BFS traversal strictly along the established edges of the Layer 0 graph. The traversal continues adding connected nodes to the group $\mathcal{G}_i$ until a predefined group capacity limit B is reached. This approach yields a structural advantage: *Topological Inlining*. Because the BFS traversal explicitly follows the graph edges, long-range connections are physically inlined into the same data block as the anchor, regardless of their actual spatial distance. Consequently, if a path in the original HNSW graph is crucial to resolving a query, the required nodes are colocated within a single retrieved block. This transforms the unpredictable pointer chasing of graph traversal into deterministic, sequential memory accesses within the target groups.

C. Overlapping Boundary Padding

In private search systems, varying block sizes leak volume patterns, allowing adversaries to infer data density and cluster distributions. To mitigate this, cryptographic storage requires all outsourced data blocks to be padded to a strict, uniform size B. Traditional systems [9] achieve this by injecting dummy data (e.g., zero-vectors) into smaller clusters, which wastes storage and network bandwidth.

HGI transforms this encrypted padding constraint into an advantage. When generating the partitions via BFS, instead of terminating early and padding with dummy data, we allow the BFS traversal to naturally over-expand across the geometric boundaries of each anchor until the fixed capacity B is filled with real graph nodes.

Algorithm 1: HGI Offline Construction

Input: Layer-0 Graph $G_0 = (\mathcal{V}_0, \mathcal{E}_0)$, Anchors E (Layer ≥ 1), Block Size B
Output: Discrete Group Set $\mathbf{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_{|E|}\}$ of uniform capacity B

$\mathbf{G} \leftarrow \{\}$;
for *each anchor* $e_i \in E$ **do**
  $Q \leftarrow$ Queue($[e_i]$), $\mathcal{G}_i \leftarrow \{e_i\}$;
  while $Q \neq \emptyset$ ***and*** $|\mathcal{G}_i| < B$ **do**
    $curr \leftarrow Q$.pop();
    for *each neighbor* n *of* $curr$ *in* G_0 **do**
      if $n \notin \mathcal{G}_i$ **then**
        $\mathcal{G}_i$.insert(n);;
        Q.push(n);
        if $|\mathcal{G}_i| == B$ **then break**;
  $\mathbf{G}$.append($\mathcal{G}_i$);
$\mathcal{V}_{\text{res}} \leftarrow \mathcal{V}_0 \bigcup_{\mathcal{G} \in \mathbf{G}} \mathcal{G}$ **for** *each orphan node* $u \in \mathcal{V}_{res}$ **do**
  $e^* \leftarrow \arg\min_{e_j \in E} \text{dist}_{G_0}(u, e_j)$ $\mathbf{G}[e^*]$.absorb(u)
for *each group* $\mathcal{G}_i \in \mathbf{G}$ **do**
  while $|\mathcal{G}_i| < B$ **do**
    $\mathcal{G}_i$.insert(SelectBoundaryNeighbor($\mathcal{G}_i, G_0$)) ;
return $\mathbf{G}$;

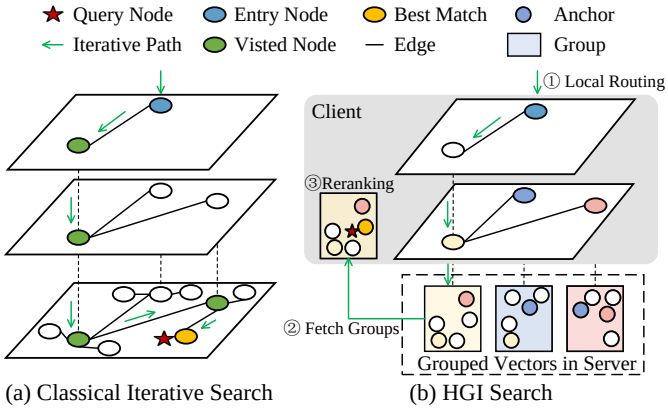
As shown in Figure 4, a node located near the boundary of two anchors, e_i and e_j, will be redundantly packed into both group $\mathcal{G}_i$ and group $\mathcal{G}_j$. This generates natural overlapping boundaries among the discrete partitions. In high-dimensional semantic spaces, query targets often lie on the boundaries of clusters. By naturally replicating boundary nodes between the near groups, HGI solves the boundary truncation effect that troubles traditional IVF indexes [9], [39], [40]. This redundancy ensures that even if the client retrieves only a limited number of target groups, the necessary neighborhood context is preserved, thereby maintaining a high recall rate comparable to that of HNSW.

D. HGI Construction and Search

To formalize the topology distillation paradigm, we outline the offline index construction and the plaintext online search.

Algorithm 1 and Figure 4 describe the offline construction of HGI. The algorithm iterates over the anchor nodes extracted from the upper layers of HNSW. For each anchor, it executes a topology-preserving BFS traversal restricted to Layer 0 edges, natively distills local topological neighborhoods and long-range edges into a fixed-size group.

Algorithm 2 and Figure 5 detail the corresponding online search. As shown in Figure 5(a), classical HNSW search suffers from data-dependent sequentiality, resulting in severe multi-RTT latency. HGI entirely decouples this data-dependent flow (Figure 5(b)). The client first performs a high-speed greedy search within the plaintext upper layers to identify the target anchors (① Local Routing). Subsequently, the client fetches the discrete blocks corresponding to these anchors



(a) Classical Iterative Search (b) HGI Search

Fig. 5: Comparison of the search process.

TABLE I: Reachability and coverage of HNSW Layer-0 nodes in plain HGI.

Dataset	Anchors	B=500	B=1K	B=2K	B=4K
SIFT-1M	31,139	99.79%	**100.00%**	100.00%	100.00%
arXiv	30,801	99.63%	99.95%	**100.00%**	100.00%
OpenAI	30,801	99.82%	99.98%	**100.00%**	100.00%
GIST-1M	31,139	93.52%	96.78%	97.90%	**100.00%**

in a single batch interaction (② Fetch Groups). Finally, the exact nearest neighbors are extracted within the client's local memory (③ Reranking). This architectural decoupling effectively reduces the latency bottleneck into a 1-RTT pipeline and improves the system throughput.

Search Completeness and Graph Coverage Guarantee. A potential concern with capacity-bounded BFS packing is whether all nodes in Layer 0 remain reachable without leaving orphan nodes. As shown in Table I, we empirically evaluate graph coverage across varying group capacities B on all four datasets. In plain BFS mode, the small-world property of HNSW ensures that nodes naturally cluster closely around upper-layer anchors: setting $B \geq 1000$ already achieves high coverage on real-world datasets. Any residual boundary nodes are seamlessly absorbed into their nearest anchor groups. HGI is able to guarantee 100.00% node coverage, proving that almost no target vector is rendered unreachable (formally proved in Appendix B).

V. Compute-Free Encrypted Filtering

While the HGI index reorganizes the dataset into navigable blocks to avoid iterative network round trips, retrieving entire blocks in their encrypted form still incurs considerable bandwidth overhead. Traditional secure filtering relies on Homomorphic Encryption to compute distances over ciphertexts, which introduces a huge computational overhead on the server. In this section, we introduce a lightweight filtering mechanism that shifts the server-side workload from heavy ciphertext computation to efficient encrypted lookups.

A. From Distance Comparison to Exact Lookup

As analyzed in § II-C, RaBitQ compresses vectors v into D-bit binary codes $\hat{v}$, converting Euclidean similarity to Hamming distance. Thus, filtering translates to finding vectors where $H(\hat{q}, \hat{v}) \leq r$. However, computing $H(\hat{q}, \hat{v})$ over ciphertexts remains expensive. We now formalize the Pigeonhole Principle from **Insight 2** to eliminate this computation. We partition $\hat{v}$ into m disjoint sub-segments: $\hat{v} = (s_1^v \| s_2^v \| \dots \| s_m^v)$, where $s_i^v \in \{0,1\}^{\frac{D}{m}}$. The total distance is the sum of sub-segment distances: $H(\hat{q}, \hat{v}) = \sum_{i=1}^{m} H(s_i^q, s_i^v)$. If $r < m$, at least one sub-segment must match exactly: $H(\hat{q}, \hat{v}) \leq r \implies \exists j \in \{1, \dots, m\}$, s.t. $H(s_j^q, s_j^v) = 0 \iff s_j^q = s_j^v$. Consequently, vectors within the Hamming radius can be precisely retrieved via combinations of sub-segment exact matches. Let $M_j(x) = \{v \mid s_j^v = x\}$ denote the set of vector IDs returned by looking up key x in the j-th sub-table. Since practical high recall is achieved with $r \leq 1$, we specify the exact lookup logic:

- $r = 0$ (Exact Match): The result R_0 is the intersection of matches across all m sub-tables using the original query sub-segments: $R_0 = \bigcap_{j=1}^{m} M_j(s_j^q)$.
- $r = 1$ (1-bit Flip): We enumerate all possibilities where exactly one bit flips in one sub-segment. Let e_k be a unit vector flipping the k-th bit within a sub-segment. We generate all D combinations of trapdoors (flipping one bit in one of the m sub-segments) and take the union of their intersection results: $R_1 = R_0 \cup \bigcup_{i=1}^{m} \bigcup_{k=1}^{\frac{D}{m}} \left(M_i(s_i^q \oplus e_k) \cap \bigcap_{j \neq i} M_j(s_j^q) \right)$.

This equivalence enables the server to perform $O(1)$ lookups to support compute-free filtering while ensuring that all vectors within the Hamming radius are retrieved.

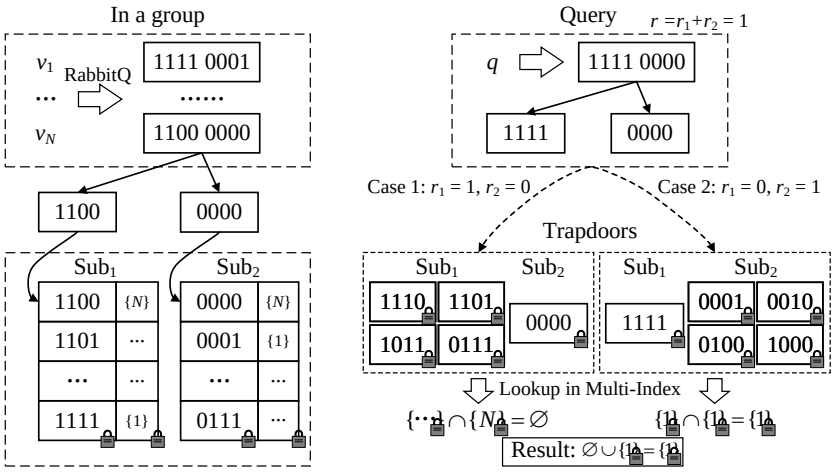
B. Group-Scoped Multi-Index

A naive implementation of the pigeonhole principle for encryption lookup involves building a global multi-index over the entire database. However, in billion-scale high-dimensional spaces, a global index suffers from a high rate of false positives. Due to hash collisions, millions of dissimilar vectors may end up sharing a single identical subsegment, forcing the server to return many irrelevant candidates and wasting network bandwidth. To address this limitation, we introduce the

Algorithm 2: HGI-based Online Retrieval

Input: Query vector q, HNSW Upper Layers $G_{\geq 1}$, Groups $\mathcal{G}$, Target Count τ, Result Size k
Output: Top-k Nearest Neighbors

$S_{\text{target}} \leftarrow \text{GreedySearch}(G_{\geq 1}, q, \tau)$;
$C = \{\}$;
for *each* $e_i \in S_{target}$ *in parallel* **do**
  $C_i \leftarrow \text{FetchGroup}(\mathcal{G}, e_i)$;
  `// O(1) Lookup`
  $C \leftarrow C \cup C_i$;
Result $= \{\}$;
for *each vector* $v \in C$ **do**
  $d \leftarrow ||q - v||^2$;
  Result.insert(v, d);
Keep the top-k candidates in Result based on d;
return Result;



(a) Group-Scoped Multi-Index (b) Trapdoor Generation and Exact Lookup

Fig. 6: A toy example of the compute-free filtering mechanism in a specific group via multi-index sub-segment matching. For clarity of presentation, all index keys, query segments, and retrieved IDs are illustrated in plaintext. In the actual system, all components are encrypted. The binary codes are divided to build independent sub-tables. During a query, the client generates trapdoors for its sub-segments, incorporating bounded bit-flip variations (r_i means the number of bit-flips in the sub-segments Sub_i), to achieve a Hamming radius $r = \sum_i^m r_i$. The server performs exact lookups across the sub-tables and computes the union of the matched Vector IDs.

Group-Scoped Multi-Index. Instead of indexing sub-segments globally, we construct an independent encrypted lookup table, essentially a *key-value* dictionary, within each topological group generated by HGI. As shown in the left part of Figure 6, for a given group, its encrypted lookup table consists of m distinct sub-tables. The construction process for the l-th sub-table is as follows:

1) Each binary vector within the group is split into m disjoint sub-segments.
2) The l-th sub-segment of the vector is calculated by PRF, and this result serves as the *key* for the l-th sub-table.
3) For all vectors in the group that share this identical PRF value, their unique IDs are encrypted and aggregated. This list of encrypted IDs is stored as the *value* corresponding to the *key*.

Furthermore, the system maintains a separate encrypted data store on the server side, where these encrypted IDs act as pointers mapping to the encrypted original float32 vectors. Because of their different functions, sub-segments, IDs, and original vectors need to be encrypted differently. As sub-segments and IDs are used for encryption lookups and comparisons, deterministic encryption (e.g., PRF) ensures that the same plaintext yields the same ciphertext. Since the original vectors do not require ciphertext comparison, we use randomized encryption (e.g., AES-CBC) to protect their confidentiality. Every Multi-index is built in a single HGI group, ensuring efficient candidate filtering.

C. Trapdoor Generation and Exact Lookup

By structuring the multi-index at the block scope, the online query execution is streamlined into a highly efficient, stateless retrieval pipeline.

Trapdoor Generation. During a query, the client identifies a set of target groups $\mathcal{G}_{target}$ through local routing over the plaintext upper layers of the HNSW graph. The client quantizes the query vector Q into a binary code and splits it into m sub-segments. The client generates C_m^r ($r = 0$ or 1) lookup trapdoors with different radius distributions in sub-segments (bit-flipping, as shown in Figure 6) and encrypts them using a PRF. The client then transmits these trapdoors, along with all of $\mathcal{G}_{target}$, to the server in a single request.

Zero-Compute Server Lookup. Upon receiving the request, the server executes exact encrypted lookups. As shown in the right panel of Figure 6, for each specified group in $\mathcal{G}_{target}$, the server probes the group's m sub-tables using the trapdoors it receives. If a trapdoor matches a *key* in the l-th sub-table, which means this key's plaintext matches the query's l-th sub-segment within the radius r_l, the server then retrieves the associated *value* (the list of encrypted IDs). For each target group, after probing its m sub-tables, the server combines the returned ID lists by r: $r = 0$ intersects the m original sub-segment matches, while $r = 1$ unions the corresponding intersections over all single-bit-flip variants. A trapdoor probes only one sub-table, and intersections are across the m sub-segment matches of the same group. Finally, using these IDs as pointers, the server fetches the corresponding encrypted original vectors and returns this refined candidate set to the client. This design decouples data retrieval from heavy cryptographic primitives.

VI. EHSearch Workflow

By combining the topology-distilled HGI index with the block-scoped encrypted multi-index, we establish the foundation for a high-throughput, low-latency private search system, Encrypted HGI-based Search (EHSearch). Figure 7 illustrates the complete architecture and the end-to-end execution flow of EHSearch, demonstrating how the decoupled data layout effectively confines the retrieval process to a single network round-trip (1-RTT).

A. Offline Construction

$\mathsf{Init}(\mathsf{DB})$: As shown in Figure 7(a), the data owner restructures the DB into topological groups (§IV-B), then generates independent Group-Scoped Multi-Indexes for each part (§V-B). These structures, encrypted via standard PRF, are outsourced to the server. The sparse upper layers of the HNSW graph are preserved on the client as a global routing cache.

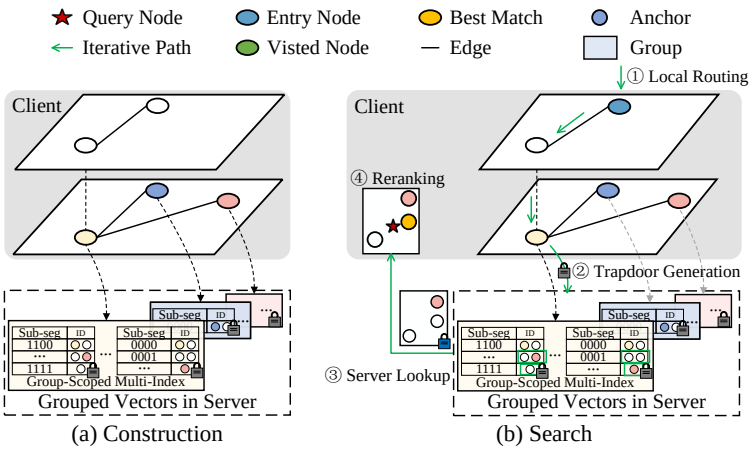


Fig. 7: System architecture and search workflow of EHSearch. (a) Construction: The distilled topological groups are outsourced to the server as encrypted Group-Scoped Multi-Indexes, while the sparse upper layers are preserved on the client. (b) Search: The end-to-end retrieval is strictly bound to a single round-trip. The client executes local routing (①) and dispatches sub-segment trapdoors (②). The server performs compute-free exact lookups (③) within the targeted multi-indexes. Finally, the client decrypts the retrieved candidate vectors for exact reranking (④) entirely in local memory.

TABLE II: Complexity comparison of private search systems.

System **Technical Path**	**Compass** [11] (HNSW+ORAM)	**Pacmann** [10] (NSW+PIR)	**Tiptoe** [9] (IVF+HE)	**EHSearch** (HGI+Exact Lookup)
Network RTTs	$O(L)$	$O(L)$ (optimized)	1	1
Bandwidth	$O(L \cdot \log N \cdot d)$	$O(L \cdot \sqrt{N} \cdot d)$	$O(\tau \cdot B \cdot d)$	$O(\tau(m+rd)+\lvert C\rvert d)$
Filtering Compute	$O(L \cdot (\log N + d))$	$O(L \cdot N \cdot d)$	$O(\tau \cdot B \cdot d)$	$O(\tau(m+rd))$
Client Storage	$O(N)$	$O(\sqrt{N})$	$O(\frac{N}{B} \cdot d)$	$O(N_{upper} \cdot d)$

B. Online Search

Illustrated in Figure 7(b), online search contains four phases:
① $\mathsf{LocRout}(Q)$: **Lightweight Client-Side Routing**. Upon receiving a query, the client performs high-speed greedy routing within its local plaintext cache to identify target groups. The client dispatches these group IDs to the server.
② $\mathsf{TrapGen}(Q, r)$: **Trapdoor Generation**. The client quantizes the query, splits it into sub-segments, and generates the corresponding cryptographic trapdoors bounded to the target groups (as in §V-C). The client dispatches these compact trapdoors to the server.
③ $\mathsf{Filter}(\text{Trapdoors}, \text{Targets})$: **Compute-Free Lookup**. Upon receiving the request, the server performs exact encrypted filtering. Leveraging the lock-free intra-query parallelization enabled by HGI (§IV-D), the server dynamically dispatches the lookup tasks to a pool of worker threads. Each thread probes the designated multi-index sub-tables within its assigned group in $O(1)$ time. If a trapdoor matches, the associated encrypted payload is retrieved. The server computes the union of the matched encrypted payloads across all targeted groups and returns them to the client within the 1-RTT network interaction. Throughout this phase, the server acts merely as a concurrent Key-Value store.
④$\mathsf{Refine}(Q, \text{Candidates})$: **Exact Client-Side Refinement**. The final phase extracts the exact Top-k results from the retrieved candidate set. Instead of relying on heuristic or quantized coarse ranking, which inherently introduces precision loss, our system allows the client to compute exact Euclidean distances directly. Supported by the filtering stage, the candidate pool is pruned from all look-up tables to a refined subset (typically a few hundred to a few thousand vectors) that easily fits in the CPU's L1/L2 cache. The client decrypts the payload (encrypted raw vectors), computes exact distances, and sorts to extract the precise Top-k results.

VII. Theoretical Analysis

We theoretically evaluate EHSearch by comparing its asymptotic complexity with that of state-of-the-art baselines. We then formalize its security under the honest-but-curious model and prove that confidentiality is strictly preserved.

A. Performance Analysis

Table II summarizes the asymptotic complexity of EHSearch compared to state-of-the-art private search systems [9], [11], [10]. Let N denote the total number of vectors in the database, and d the vector dimensionality. For iterative graph traversal, let L denote the average path length (number of hops). For IVF and EHSearch, let τ denote the number of targeted clusters/groups retrieved per query, and let B denote the block capacity. For EHSearch, the number of nodes that Layer ≥ 1 is N_{upper}, which is usually only $1\% \sim 3\%$ of N, and m is the number of sub-segments used in the multi-index. Due to space constraints, we show a full analysis in §C.

B. Security Proof Sketch of Theorem 1

We prove $\mathcal{L}$-adaptive security by constructing a PPT simulator $\mathcal{S}$ solely from the leakage profile $\mathcal{L}$. Given $\mathcal{L}^{Setup} = N$, $\mathcal{S}$ allocates $t = \lceil N/B \rceil$ dummy groups of uniform capacity B, perfectly mirroring the storage volume without leaking data density. For queries, $\mathcal{S}$ generates uniformly random strings as trapdoors consistent with $\mathsf{SearchPattern}$, and dynamically programs the simulated multi-index sub-tables to return the dummy identifiers dictated by $\mathsf{AccessPattern}$. Because raw vectors are protected under IND-CPA-secure AES-CBC, trapdoors are masked by a cryptographically secure PRF, and no

continuous metric distances are revealed, no polynomial-time adversary can distinguish the simulated transcript from a real execution. The complete sequence of hybrid games is detailed in Appendix A.

VIII. Implementation

We implemented EHSearch as a self-contained system comprising approximately 3,300 lines of C++ code[1], split into an offline index builder and an online concurrent search engine. Offline, EHSearch builds the HNSW graph, binarizes vectors with RaBitQ [25] into m sub-segments, fills the block-scoped multi-indexes, and writes AES-CBC-encrypted payloads, so no plaintext embedding is retained. Each $(group, segment)$ table is one ascending array of packed $key|id$ words, so a lookup costs a single binary search, and a per-segment *keymin/keymax* bound rejects misses without entering it. All cryptographic primitives are hardware-accelerated: AES-NI accelerates key scheduling and encryption, VAES decrypts four blocks per instruction, and an AES-based PRF derives lookup keys and trapdoors. Hot compute kernels are vectorized with FMA and dynamically dispatched to AVX-512 or AVX2 at runtime. *1) In-Memory.* For million-scale datasets, multi-indexes and payloads reside entirely in RAM. Group filtering writes into per-group private buckets, which are merged via an epoch-stamped visited array, so the query path has no locks or atomics. Queries are processed concurrently across OpenMP threads. On the client, exact Euclidean reranking is vectorized via SIMD, using `std::nth_element` to extract top-k results in $O(|C|)$ time. *2) Out-of-Core SSD.* To scale to billion-scale datasets that exceed RAM (Appendix D), the server maps the prepared ciphertext file read-only into virtual memory and issues no explicit `read` system calls; the OS handles page faults. Records are stored in ascending ID order, so matched candidate IDs are globally sorted before the payload is touched, converting scattered lookups into a single monotone sequential sweep. Three further techniques act on this path:

(i) No intermediate copy. The client decrypts each record in place, inside the mapped file, following the same ascending-ID sweep. At τ=60, this removes up to 11.5 MB of memory traffic per query. Under 32 concurrent queries, where DRAM bandwidth is the shared limit, throughput rises by 1.4–2.1×, while single-thread latency is unchanged. *(ii) Software prefetch.* Both access patterns are data-dependent and defeat hardware prefetchers, so prefetches are issued six records ahead on the sweep, and eight vectors ahead on the exact-distance rescan, lowering compute latency by 12–24 %. *(iii) Batched decryption.* Record decoding runs on VAES, four CBC blocks per instruction. The sweep reads a 4 KiB page per 528 B record, a 7.4× amplification, and the volume actually retrieved from the device is profiled via process I/O counters rather than modeled. Parallelizing within a single query does not pay off on SSD: the nested parallel region makes a query up to 2× slower at τ=5, so concurrency is applied only across queries. At this extreme scale, we additionally employ memory-bounded topology approximation (Appendix E).

[1] anonymous.4open.science/r/EHSearch

IX. Evaluation

In this section, we aim to answer the following questions:

- **Parameter Sensitivity:** How do index parameters affect the system performance? (§IX-B)
- **Comparison with State-of-the-Art:** How does EHSearch compare to state-of-the-art baselines in terms of throughput, recall, bandwidth, storage footprint, and network latency across LAN and WAN? (§IX-C)
- **In-Depth Latency and Plaintext Parity:** What is the fine-grained latency breakdown of EHSearch across execution stages, how robust is its tail latency, and how closely does it approach plaintext HNSW? (§IX-D)
- **Billion-Scale Out-of-Core Scalability:** Can EHSearch scale to larger datasets on a single SSD, even up to billion-points, and maintain practicality? (§IX-E)

A. Setup

The experiments are conducted on a personal computer with an AMD Ryzen 9 9950X3D CPU, 32GB of RAM, and a ZHITAI TiPlus7100 1 TB drive. We use real-world datasets of varying scales and dimensions, SIFT-1M∼1B [41] (N = 1,000,000 ∼ 1,000,000,000, $d = 128$), GIST-1M [41] (N = 1,000,000, $d = 960$), arXiv [42] (N = 2,253,000, $d = 768$), and OpenAI [43] (N = 999,000, d = 1536). We evaluate systems under two network profiles: (1) **LAN**, configured with 3 Gbps bandwidth and 1 ms round-trip time (RTT); and (2) **WAN**, configured with 400 Mbps bandwidth and 80 ms RTT. We compare against three baselines based on different indexes:

- **Tiptoe** [9]: The state-of-the-art single-round scheme[2] relying on IVF and HE.
- **Compass** [11]: The state-of-the-art multi-round scheme[3] utilizing HNSW over Ring ORAM [15].
- **Pacmann** [10]: The multi-round scheme[4] utilizing NSW over client-preprocessing PIR, in which PIR status must be updated after each query.

B. Parameter Sensitivity Analysis

EHSearch's compute-free filtering and topology-distilled retrieval are governed by four core parameters: the sub-segment count m, group capacity B, Hamming radius $r \in \{0, 1\}$, and the number of retrieved groups τ. Figure 8 illustrates their trade-offs against Recall@10 and QPS on SIFT-1M: *1) Hamming Radius (r) and Sub-segments (m).* As shown in Figure 8(a), r determines the filtering strictness. Setting $r = 0$ delivers low candidate pools and lightning-fast throughput, making it optimal for latency-critical applications targeting moderate accuracy. Setting $r = 1$ exponentially expands the candidate pool $|C|$, enabling EHSearch to scale to the ultra-high-accuracy regime (Recall@10 > 0.99) and match plaintext HNSW. Under $r = 1$, decreasing sub-segment width increases collision rates and expands candidate volume, which raises the recall ceiling at the cost of additional client-side Euclidean

[2] github.com/ahenzinger/tiptoe
[3] github.com/Clive2312/compass
[4] github.com/wuwuz/Pacmann

TABLE III: Comprehensive end-to-end performance comparison across datasets. Values in parentheses indicate EHSearch's speedup over the baseline.

Dataset	Scheme	QPS	Comm. (MB)	Server (GB)	Client (MB)	RTT	LAN Latency (ms)	WAN Latency (ms)
SIFT-1M	**EHSearch**	**1650.2**	**4.14**	1.42	26.4	**1**	**13.7**	**164.4**
	Compass	1.4 (1,178.7×)	9.02	6.70	32.3	8	32.6 (2.4×)	860.5 (5.2×)
	Pacmann	10.1 (163.4×)	37.60	0.58	270.2	16	116.3 (8.5×)	2032.0 (12.4×)
	Tiptoe	0.034 (48,535.3×)	736.96	15.63	0.71	1	1966.2 (143.4×)	14819.2 (90.2×)
arXiv	**EHSearch**	**147.2**	**46.71**	5.60	105.9	**1**	**140.6**	**1029.1**
	Compass	0.12 (1,226.2×)	276.75	39.74	193.8	9	746.5 (5.3×)	6215.0 (6.0×)
	Pacmann	0.866 (170.0×)	1153.12	3.45	1620.9	16	3091.0 (22.0×)	24342.4 (23.7×)
	Tiptoe	0.0029 (50,490.9×)	22601.15	92.74	4.26	1	60270.7 (428.7×)	452103.0 (439.3×)
GIST-1M	**EHSearch**	**50.9**	**113.28**	10.55	131.0	**1**	**334.9**	**2377.3**
	Compass	0.049 (1,029.7×)	345.94	50.22	242.3	9	931.0 (2.8×)	7598.7 (3.2×)
	Pacmann	0.357 (142.7×)	1441.40	4.37	2026.2	16	3859.7 (11.5×)	30108.0 (12.7×)
	Tiptoe	0.0012 (42,400.4×)	28251.44	117.22	5.33	1	75338.2 (225.0×)	565108.8 (237.7×)
OpenAI	**EHSearch**	**349.0**	**15.81**	11.18	210.0	**1**	**52.3**	**405.3**
	Compass	0.354 (986.7×)	553.50	79.47	387.6	9	1484.5 (28.4×)	11750.0 (29.0×)
	Pacmann	2.55 (136.8×)	2306.24	6.91	3241.8	16	6166.0 (117.8×)	47404.8 (117.0×)
	Tiptoe	0.0086 (40,628.5×)	45202.30	185.48	8.52	1	120540.5 (2303.0×)	904126.1 (2230.8×)

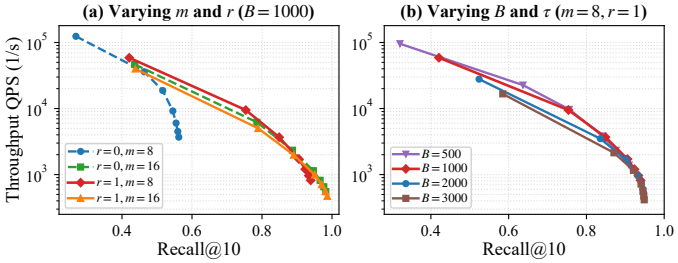


Fig. 8: Parameter sensitivity of EHSearch on SIFT-1M.

reranking. *2) Group Capacity (B) and Scanned Groups (τ).* Figure 8(b) reveals the interplay between group size and retrieval depth. A larger group capacity B ($B \geq 1000$) can encapsulate more long-range shortcut edges within a single data block, thereby lifting the achievable recall bound. Sweeping τ from 1 to 50 smoothly traverses the Pareto frontier. Crucially, at $B = 1000$ and $\tau = 20$, EHSearch already reaches Recall@10 $= 0.906$ at 1,650.2 QPS, achieving near-optimal convergence with minimal bandwidth. Due to space limits, the exhaustive parameter benchmark on all four datasets is deferred to Appendix F.

C. End-to-End Search Performance

Table III presents the comprehensive benchmark comparing EHSearch against state-of-the-art private search schemes across four real-world datasets. Following Compass [11], we evaluate at target Recall@10 ≈ 0.90 (and ≈ 0.81 on the high-dimensional OpenAI).

Throughput Superiority. EHSearch outperforms all baselines by 2 to 5 orders of magnitude. Specifically, under a continuous streaming query workload that fully saturates the serving pipeline, EHSearch achieves 1650.2 QPS on SIFT-1M and 147.2 QPS on the 768-dimensional LLM dataset arXiv, delivering a $163\times \sim 170\times$ speedup over Pacmann, $1{,}178\times \sim 1{,}226\times$ over Compass, and over $48{,}000\times$ over Tiptoe. Crucially, on the ultra-high-dimensional OpenAI dataset ($D = 1536$), EHSearch maintains a dominant throughput of 349.0 QPS. This directly refutes the conjecture that our compute-free filtering degrades for high-dimensional vectors: by pruning candidate pools via block-scoped multi-indexes, EHSearch scales gracefully without incurring dimension-induced bottlenecks in ciphertext multiplication. In contrast, Tiptoe flatlines at 0.0086 QPS due to dense homomorphic matrix multiplications, while Compass degrades to 0.35 QPS under serialized ORAM tree lookups.

Bandwidth and Storage Overhead. As shown in Table III, EHSearch achieves minimal network communication, requiring only 4.14 MB on SIFT-1M and 15.81 MB on OpenAI per query. This represents a $2.2\times \sim 35\times$ bandwidth reduction compared to Compass, $9\times \sim 146\times$ compared to Pacmann, and up to $2{,}859\times$ compared to Tiptoe (which transfers tens of gigabytes of HE ciphertexts). On the server side, EHSearch introduces modest storage amplification (merely expanding raw payloads to 11.18 GB on OpenAI), avoiding the 185.48 GB HE ciphertext bloat in Tiptoe and 79.47 GB ORAM tree overhead in Compass. On the client side, EHSearch caches only the sparse upper layers of HNSW, requiring a compact memory footprint of 26.4 MB to 210.0 MB, whereas Pacmann's preprocessing PIR consumes 3.24 GB.

Network Latency: The 1-RTT Advantage. Table III quantifies end-to-end response times across network profiles. In LAN, EHSearch already outperforms Compass by $2.4\times \sim 28.4\times$ and Pacmann by $8.5\times \sim 117.8\times$. In WAN (80ms RTT), this gap widens dramatically: multi-round interactive schemes suffer catastrophic RTT delays, with Compass surging to 11.75s ($29.0\times$ slower) and Pacmann to 47.40s ($117.0\times$ slower) on OpenAI. In sharp contrast, EHSearch strictly guarantees 1-RTT retrieval, completing queries in merely 164.4 ms on SIFT-1M and 405.3 ms on OpenAI, thereby eliminating the interactive network bottleneck in geo-distributed settings.

D. Latency Decomposition and Tail Robustness

To rigorously assess the practicality of the service, Table IV details the latency breakdown of EHSearch, along with its tail latency distributions over LAN and WAN.

Execution Breakdown. As shown in Table IV, the end-to-end computational workload is dominated by server-side multi-

TABLE IV: Latency decomposition and tail latency profile of EHSearch across datasets at target Recall@10 ≈ 0.90.

Dataset	Compute Breakdown (Mean, ms)						LAN Latency (ms)			WAN Latency (ms)		
	Route	TrapGen	Lookup	Decrypt	Rerank	Total	P50	P95	P99	P50	P95	P99
SIFT-1M	0.08	0.004	1.37	0.15	0.09	**1.69**	14.21	20.66	22.35	168.3	211.7	223.3
arXiv	0.51	0.077	10.33	3.01	1.11	**15.03**	149.08	177.19	186.56	1091.3	1285.1	1343.3
GIST-1M	0.42	0.107	20.30	8.08	2.92	**31.83**	367.13	428.36	444.85	2605.1	3050.7	3158.6
OpenAI	0.82	0.282	6.89	0.86	0.33	**9.19**	52.41	78.05	90.38	406.8	585.9	669.5

index filtering, while client routing and reranking overheads remain negligible. On SIFT-1M, client-side routing on the cached upper layers takes merely 0.08 ms, trapdoor generation takes 4 μs, and client decryption and exact reranking take 0.24 ms combined. The server exact lookup completes in 1.37 ms, yielding a total computational delay of just 1.69 ms. Even on the 1536-dimensional OpenAI, total compute time remains bounded at 9.19 ms.

Tail Latency Stability. Unlike ORAM-based systems, whose access latency varies widely due to periodic tree evictions, EHSearch ensures deterministic execution. Under LAN conditions, EHSearch achieves a P95 tail latency of 20.66 ms on SIFT-1M and 78.05 ms on OpenAI. The gap between P50 and P99 is remarkably narrow (e.g., within 8.1 ms on SIFT-1M and within 38.0 ms on OpenAI), demonstrating that our compute-free filtering avoids long-tail stragglers.

Recall vs. Latency and Plaintext Limits. Figure 9 plots end-to-end query latency versus Recall@10 across all datasets under LAN and WAN, benchmarked against plaintext HNSW. EHSearch approaches the peak recall of unencrypted HNSW, confirming that HGI preserves graph navigability with negligible accuracy degradation. Furthermore, query latency scales smoothly and monotonically, without erratic ORAM eviction spikes, as the target group counts τ increases. Crucially, transitioning to WAN shifts the Pareto frontiers horizontally by approximately 80 ms, strictly matching the single-RTT physical floor with zero multi-hop protocol inflation.

E. Billion-Scale Out-of-Core Evaluation

To evaluate scalability under memory constraints, we deploy EHSearch on an NVMe SSD backend, scaling from 1M to 1 billion vectors ($N = 1\text{M} \sim 1\text{B}$) on SIFT-1B. While prior private search evaluations top out at 100 million vectors [10], this evaluation tests the system across three orders of magnitude on commodity hardware.

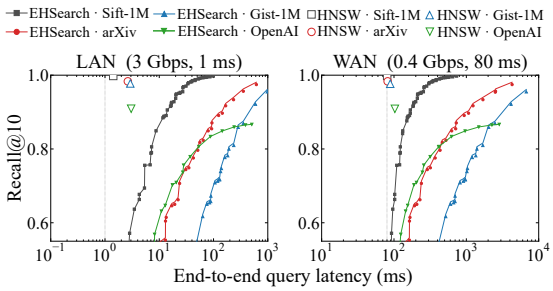


Fig. 9: End-to-end query average latency vs. Recall@10 across four datasets under LAN and WAN.

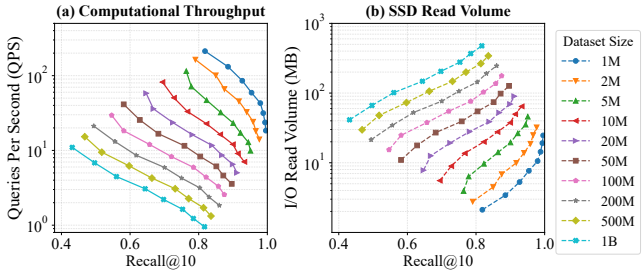


Fig. 10: Out-of-core scalability and I/O costs on NVMe SSD across 1M to 1B datasets: (a) Computational throughput (QPS) versus Recall@10; (b) Physical SSD read volume (MB) per query versus Recall@10.

Figure 10 shows throughput (QPS) and SSD read volume (MB) versus Recall@10. On 1M vectors, EHSearch achieves $18.4 \sim 212.8$ QPS with Recall@10 reaching up to 99.6%, reading $2.1 \sim 24.7$ MB per query. As the database scales to 1B vectors, the system yields $0.95 \sim 11.0$ QPS, achieving up to 81.7% Recall@10. Figure 10(b) shows that throughput degradation across scales correlates directly with physical I/O: achieving 81.7% recall at 1B requires reading 476.0 MB of encrypted payloads from disk.

X. Related Work

Our work sits at the intersection of high-dimensional vector indexing and cryptographic systems. We now review existing plaintext index architectures and the trade-offs of existing private search systems.

Plaintext Vector Indexes. The evolution of ANNS has produced highly optimized structures, generally categorized into LSH-based [44], [45], [46], [47], [48], tree-based [49], [50], [51], [52], [53], [54], [55], [56], [57], quantization-based [58], [59], [60], [25], and graph-based [61], [62], [63], [21], [64], [65], [66], [67] methods. Specifically, graph indexes such as HNSW have become the industry standard for in-memory retrieval due to their high recall. However, their reliance on data-dependent pointer chasing makes them notoriously hostile to environments with high memory-access latency (e.g., SSDs or network storage). While these plaintext techniques achieve sublinear latency, their underlying data layouts are incompatible with cryptographic constraints.

Iterative Private Search. To preserve the high recall of graph-based indexes in secure environments, several systems directly wrap the graph structure with cryptographic primitives. Systems that rely on ORAM or Multi-Party Computation, such as Compass [11], Panther [39], and other multi-server protocols [68], [69], [40], aim to hide the traversal path. Similarly,

Pacmann [10] utilizes client-preprocessing PIR to secure graph routing. However, these systems universally suffer from the sequential dependency of graph walks. Resolving a query inherently entails multiple network round-trips and sequential cryptographic evaluations, which introduce prohibitive latency in Wide Area Networks.

Single-Round Private Search. To eliminate the iterative RTT bottleneck, an alternative paradigm adopts flat IVF indexes to realize single-shot batch retrieval. Homomorphic clustering schemes like Tiptoe [9] and Wally [18] avoid multi-round network delays. However, they rely on hard geometric partitioning, which severs topological connections and forces the client to fetch many padded clusters to maintain reasonable recall. Furthermore, fetching these clusters requires dense matrix multiplications over Homomorphic Encryption (HE) [70], [71], thereby making the server a severe computational bottleneck.

XI. Conclusion

This paper presented EHSearch, a practical private vector search system that breaks the dependency on heavy cryptographic machinery. HGI converts dynamic graph walks into deterministic 1-RTT retrieval with verified graph coverage, while Compute-Free Encrypted Filtering replaces heavy cryptography with lightweight $O(1)$ PRF lookups. Evaluations across image and LLM text embeddings demonstrate that EHSearch outperforms state-of-the-art private baselines by 2 to 5 orders of magnitude in throughput, preserves sub-second WAN latency, and scales to 1 billion vectors on a single SSD.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[3] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu *et al.*, "Milvus: A purpose-built vector data management system," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2614–2627.

[4] F. Li, "Cloud-native database systems at alibaba: Opportunities and challenges," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2263–2272, 2019.

[5] Z. Niu, X. Tian, X. Peng, and X. Chen, "Blendhouse: A cloud-native vector database system in bytehouse," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2025, pp. 4332–4345.

[6] X. Pan, M. Zhang, S. Ji, and M. Yang, "Privacy risks of general-purpose language models," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1314–1331.

[7] C. Song and A. Raghunathan, "Information leakage in embedding models," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 377–390.

[8] K. Chelani, T. Sattler, F. Kahl, and Z. Kukelova, "Privacy-preserving representations are not enough: Recovering scene content from camera poses," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 132–13 141.

[9] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich, "Private web search with tiptoe," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 396–416.

[10] M. Zhou, E. Shi, and G. Fanti, "Pacmann: Efficient private approximate nearest neighbor search," in *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[11] J. Zhu, L. Patel, M. Zaharia, and R. A. Popa, "Compass: Encrypted semantic search with high accuracy," in *19th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2025*. USENIX Association, 2025, pp. 915–938.

[12] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM computing surveys (CSUR)*, vol. 38, no. 2, pp. 6–es, 2006.

[13] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1247–1260, 2015.

[14] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: an extremely simple oblivious ram protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[15] L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas, "Constants count: Practical improvements to oblivious RAM," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. USENIX Association, 2015, pp. 415–430.

[16] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[17] M. Zhou, A. Park, W. Zheng, and E. Shi, "Piano: extremely simple, single-server pir with sublinear server computation," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4296–4314.

[18] H. Asi, F. Boemer, N. Genise, M. H. Mughees, T. Ogilvie, R. Rishi, K. Talwar, K. Tarbe, A. Wadia, R. Zhu *et al.*, "Scalable private search with wally," *arXiv preprint arXiv:2406.06761*, 2024.

[19] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012.

[20] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[21] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[22] K. J. Supowit, "The relative neighborhood graph, with an application to minimum spanning trees," *Journal of the ACM (JACM)*, vol. 30, no. 3, pp. 428–448, 1983.

[23] J. W. Jaromczyk and G. T. Toussaint, "Relative neighborhood graphs and their relatives," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1502–1517, 2002.

[24] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2025.

[25] J. Gao and C. Long, "Rabitq: quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–27, 2024.

[26] W. Huang, X. Wang, D. Zhu, X. Ma, and J. Ma, "When multimedia meets security: Privacy-preserving cross-modal retrieval for large-scale data," *IEEE Transactions on Information Forensics and Security*, vol. 21, pp. 319–332, 2026.

[27] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 1156–1167.

[28] C. Gong, X. Wang, D. Zhu, C. Huang, Z. Ma, and J. Ma, "Enabling efficient and privacy-preserving sequence similarity query on encrypted genomes," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 6, pp. 6058–6073, 2025.

[29] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (J. ACM)*, vol. 33, no. 4, p. 792–807, 1986.

[30] M. Damie, F. Hahn, and A. Peter, "A highly accurate Query-Recovery attack against searchable encryption using Non-Indexed documents," in *30th USENIX Security Symposium*, 2021, pp. 143–160.

[31] H. Nie, W. Wang, P. Xu, X. Zhang, L. T. Yang, and K. Liang, "Query recovery from easy to hard: Jigsaw attack against SSE," in *33rd USENIX Security Symposium*, 2024, pp. 2599–2616.

[32] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Data recovery on encrypted databases with k-nearest neighbor query leakage," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1033–1050.

[33] ——, "The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1223–1240.

[34] P. Li, H. Chen, K. Liang, and E. A. Markatou, "Learning from leakage: Database reconstruction from just a few multidimensional range queries," in *33rd Annual Network and Distributed System Security*

Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026, 2026.

[35] Z. Gui, K. G. Paterson, and S. Patranabis, "Rethinking searchable symmetric encryption," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1401–1418.

[36] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2020.

[37] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *Advances in neural information processing Systems*, vol. 32, 2019.

[38] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM computing surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[39] J. Li, Z. Huang, M. Zhang, C. Hong, J. Liu, T. Wei, and W. Chen, "Panther: Private approximate nearest neighbor search in the single server setting," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. New York, NY, USA: ACM, 2025, p. 365–379.

[40] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi, "{SANNS}: Scaling up secure approximate {k-Nearest} neighbors search," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2111–2128.

[41] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[42] L. Kuffo, E. Krippner, and P. Boncz, "Pdx: A data layout for vector similarity search," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–26, 2025.

[43] Kumar Shivendu and Nirant Kasliwal, "dbpedia-entities-openai-1m," 2023. [Online]. Available: https://huggingface.co/datasets/KShivendu/dbpedia-entities-openai-1M

[44] A. Andoni and I. Razenshteyn, "Optimal data-dependent hashing for approximate near neighbors," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, 2015, pp. 793–801.

[45] Y. Lei, Q. Huang, M. Kankanhalli, and A. K. Tung, "Locality-sensitive hashing scheme based on longest circular co-substring," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2589–2599.

[46] Y. Tian, X. Zhao, and X. Zhou, "Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 3, pp. 1000–1015, 2023.

[47] J. Wei, B. Peng, X. Lee, and T. Palpanas, "Det-lsh: a locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search," *arXiv preprint arXiv:2406.10938*, 2024.

[48] B. Zheng, Z. Xi, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen, "Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search," *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 643–655, 2020.

[49] M. Chatzakis, P. Fatourou, E. Kosmas, T. Palpanas, and B. Peng, "Odyssey: A journey in the land of distributed data series similarity search," *arXiv preprint arXiv:2301.11049*, 2023.

[50] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas, "Coconut: A scalable bottom-up approach for building data series indexes," *arXiv preprint arXiv:2006.13713*, 2020.

[51] M. Linardi and T. Palpanas, "Scalable, variable-length similarity search in data series: The ulisse approach," *Proceedings of the VLDB Endowment*, vol. 11, no. 13, pp. 2236–2248, 2018.

[52] B. Peng, P. Fatourou, and T. Palpanas, "Messi: In-memory data series indexing," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 337–348.

[53] ——, "Sing: Sequence indexing using gpus," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 1883–1888.

[54] Q. Wang, I. Ileana, and T. Palpanas, "Leafi: Data series indexes on steroids with learned filters," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–27, 2025.

[55] Q. Wang and T. Palpanas, "Deep learning embeddings for data series similarity search," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 1708–1716.

[56] Z. Wang, Q. Wang, P. Wang, T. Palpanas, and W. Wang, "Dumpy: A compact and adaptive index for large data series collections," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–27, 2023.

[57] ——, "Dumpyos: A data-adaptive multi-ary index for scalable data series similarity search," *The VLDB Journal*, vol. 33, no. 6, pp. 1887–1911, 2024.

[58] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 6, pp. 1247–1260, 2014.

[59] M. Norouzi and D. J. Fleet, "Cartesian k-means," in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, 2013, pp. 3017–3024.

[60] Y. Xia, K. He, F. Wen, and J. Sun, "Joint inverted indexing," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3416–3423.

[61] I. Azizi, K. Echihabi, and T. Palpanas, "Elpis: Graph-based similarity search for scalable data science," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1548–1559, 2023.

[62] C. Fu, C. Wang, and D. Cai, "High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4139–4150, 2021.

[63] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *arXiv preprint arXiv:1707.00143*, 2017.

[64] J. V. Munoz, M. A. Gonçalves, Z. Dias, and R. d. S. Torres, "Hierarchical clustering-based graphs for large scale approximate nearest neighbor search," *Pattern Recognition*, vol. 96, p. 106970, 2019.

[65] X. Zhao, Y. Tian, K. Huang, B. Zheng, and X. Zhou, "Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces," *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 1979–1991, 2023.

[66] Z. Yin, J. Gao, P. Balsebre, G. Cong, and C. Long, "Deg: Efficient hybrid vector search using the dynamic edge navigation graph," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–28, 2025.

[67] I. Azizi, K. Echihabi, and T. Palpanas, "Graph-based vector search: An experimental evaluation of the state-of-the-art," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–31, 2025.

[68] S. Servan-Schreiber, S. Langowski, and S. Devadas, "Private approximate nearest neighbor search with sublinear communication," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 911–929.

[69] S. Song, L. Liu, R. Chen, W. Peng, and Y. Wang, "Secure approximate nearest neighbor search with locality-sensitive hashing," in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 411–430.

[70] J. J. Engelsma, A. K. Jain, and V. N. Boddeti, "Hers: Homomorphically encrypted representation search," *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 4, no. 3, pp. 349–360, 2022.

[71] S. Bian, H. Pan, J. Hu, Z. Zhang, Y. Fu, J. Hua, Y. Chen, B. Zhang, Y. Jin, J. Dong *et al.*, "Engorgio: An arbitrary-precision unbounded-size hybrid encrypted database via quantized fully homomorphic encryption," *Cryptology ePrint Archive*, 2025.

[72] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *International conference on database theory*. Springer, 1999, pp. 217–235.

[73] G. Fuchsbauer, R. Ghosal, N. Hauke, and A. O'Neill, "Approximate distance-comparison-preserving symmetric encryption," in *International Conference on Security and Cryptography for Networks*. Springer, 2022, pp. 117–144.

[74] Y. Liu, Y. Zhang, J. Xie, H. Li, J. X. Yu, and J. Cui, "Privacy-preserving approximate nearest neighbor search on high-dimensional data," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 2025, pp. 3017–3029.

APPENDIX A
SECURITY PROOF

We formally prove the security of EHSearch using the standard real-vs-ideal simulation paradigm. We show that for any PPT adversary $\mathcal{A}$, we can construct a PPT simulator $\mathcal{S}$ such that the real execution view is computationally indistinguishable from the simulated view.

Theorem 1 (Restated). *EHSearch is $\mathcal{L}$-adaptively secure if AES-CBC is an IND-CPA-secure encryption scheme and the PRF is cryptographically secure.*

A. Simulator Construction

Given $\mathcal{L}(\mathsf{DB}, \vec{q})$, the simulator $\mathcal{S}$ works as follows:

Simulate Init**:** Let $t = \lceil \frac{N}{B} \rceil$. For each group $j = 1..t$, $\mathcal{S}$ allocates a list of B ciphertexts, each being a uniformly random byte string of length equal to the AES-CBC encryption of a raw vector. Because all groups are strictly padded to the exact same byte length by the overlapping boundary padding mechanism, the simulated storage volume pattern is identical to the real one, leaking no data density information. For each sub-table $T_{j,\ell}$ in the Group-Scoped Multi-Index, $\mathcal{S}$ creates an empty dictionary.

Simulate $\mathsf{LocRout}$ & $\mathsf{TrapGen}$**:** $\mathcal{S}$ maintains a map $\mathsf{TrapMap} : \mathsf{QueryID} \to \{0,1\}^\lambda$. For each query q_i, if it is indicated equality with a previous query q_j by $\mathsf{SearchPattern}(q_i, q_j)$, $\mathcal{S}$ sets the trapdoor vector $\tau_i = \tau_j$. Otherwise, $\mathcal{S}$ chooses m random strings $\tau_i \leftarrow \{0,1\}^\lambda$ uniformly at random. $\mathcal{S}$ outputs τ_i alongside the target group identifiers provided by $\mathsf{AccessPattern}$.

Simulate Filter**:** When a query q_i is issued, $\mathcal{S}$ extracts the exact set of target group identifiers and the set of matched vector identifiers from $\mathsf{AccessPattern}(q_i)$. $\mathcal{S}$ programs the simulated empty sub-tables on the fly such that the simulated trapdoors map to the encrypted dummy IDs specified in $\mathsf{AccessPattern}(q_i)$. $\mathcal{S}$ returns these dummy encrypted IDs and their corresponding random ciphertext payloads to the adversary.

B. Indistinguishability Proof

Proof of Theorem 1. We prove that $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) \approx_c \mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ via a sequence of hybrid experiments.

Hybrid H_0**:** The real world experiment $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$.

Hybrid H_1**:** In this hybrid, we replace the AES-CBC encrypted payloads (the raw vectors) with uniformly random byte strings of the same length, and replace the PRF-generated sub-table keys with uniformly random strings. The search behavior remains unchanged. Because AES-CBC is an IND-CPA secure pseudorandom permutation, the views in H_0 and H_1 are computationally indistinguishable. Crucially, the uniform block size B ensures that the groups' byte-length distributions remain identical, preventing any storage volume-based distinctions.

Hybrid H_2**:** In this hybrid, we replace the query trapdoors (the PRF values of the query sub-segments) with uniformly random bit strings, maintaining the equality pattern dictated by $\mathsf{SearchPattern}$. If q_i and q_j have the same search pattern, the same random string is used; otherwise, fresh random strings are drawn. Since the trapdoors in H_1 are outputs of PRF, replacing them with uniform random strings (conditioned on the equality pattern) results in a computationally indistinguishable view. Thus, $H_1 \approx_c H_2$.

Hybrid H_3**:** In this hybrid, we replace the real Group-Scoped Multi-Index construction and the Compute-Free Server Lookup mechanism with the simulated ones. Instead of inserting real key-value pairs derived from RaBitQ binary codes during the Offline Construction, $\mathcal{S}$ keeps the sub-tables empty initially. When a query q_i is issued, $\mathcal{S}$ dynamically programs the simulated sub-tables on the fly such that the trapdoors map to the encrypted dummy IDs specified in $\mathsf{AccessPattern}(q_i)$. Because the keys in H_2 are already uniformly random (from the PRF replacement), the adversary cannot distinguish between real pre-computed table entries and dynamically programmed entries. Furthermore, the returned result set is enforced to exactly match the $\mathsf{AccessPattern}$ leakage. Therefore, $H_2 \approx_c H_3$.

Hybrid H_4**:** This is the ideal world experiment $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$. By inspection, H_3 is exactly the ideal world, as the simulator only relies on the leakage $\mathcal{L}(\mathsf{DB}, \vec{q})$ to generate the view.

By the hybrid argument, we conclude:

$$\left|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

□

C. Hardness Analysis of High-Dimensional Embedding Reconstruction

In this section, we formalize why bounded pattern leakage ($\mathsf{AccessPattern}, \mathsf{SearchPattern}$) under standard symmetric encryption does not allow an honest-but-curious adversary to reconstruct continuous high-dimensional embeddings $\mathcal{D} = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\} \subset \mathbb{R}^D$ ($D \geq 100$).

1. Orthogonal Isometry and Continuous Rotational Ambiguity. Unlike Order-Preserving Encryption (OPE) or Distance-Recoverable Encryption (DRE) schemes that reveal metric distances or pairwise orderings, EHSearch uses standard IND-CPA AES-CBC and PRFs, revealing strictly *zero* continuous distances, inner products, or metric coordinates. In $\mathbb{R}^D$, any orthogonal transformation $\mathbf{U} \in O(D)$ satisfies $\|\mathbf{U}\mathbf{v}_i - \mathbf{U}\mathbf{v}_j\| = \|\mathbf{v}_i - \mathbf{v}_j\|$ and $\langle \mathbf{U}\mathbf{v}_i, \mathbf{U}\mathbf{v}_j \rangle = \langle \mathbf{v}_i, \mathbf{v}_j \rangle$. Because RaBitQ binarization applies randomized orthogonal rotations over unit-normalized embeddings, simultaneously rotating all database vectors and queries by $\mathbf{U}$ preserves all Hamming distances, all sub-segment pigeonhole matches, and produces an identical $\mathsf{AccessPattern}$ transcript. The orthogonal Lie group $O(D)$ has continuous rotational degrees of freedom:

$$\dim(O(D)) = \frac{D(D-1)}{2}. \tag{3}$$

For $D = 128$, this introduces 8,128 continuous unknown rotational parameters (and over 1.17×10^6 for $D = 1536$). Without at least $D + 1$ non-coplanar known plaintext anchor vectors to fix the global coordinate frame, unique coordinate reconstruction is information-theoretically indeterminate.

2. Unordered Set Degeneracy (Infinite Valid Solutions). State-of-the-art geometric reconstruction attacks [32], [33] infer coordinates by modeling kNN access overlaps as systems of intersecting Voronoi hyperplanes. However, as proven in [32], when queries reveal only unordered, unranked identifier sets without relative distances, the constraints define open polyhedral cones in $\mathbb{R}^{N \times D}$. The set of satisfying coordinate configurations is a nonempty open set containing an uncountable number of valid solutions that yield the exact same access transcript.

3. Measure Concentration and Hadamard Instability. By the distance concentration phenomenon in high-dimensional spaces [72], under mild distributional assumptions:

$$\lim_{D\to\infty} \frac{\mathrm{Var}(\|\mathbf{v}_i - \mathbf{v}_j\|_2)}{\mathbb{E}[\|\mathbf{v}_i - \mathbf{v}_j\|_2]^2} = 0 \implies \frac{D_{\max} - D_{\min}}{D_{\min}} \xrightarrow{P} 0. \quad (4)$$

As D grows large, the relative distance contrast between the nearest and the farthest vectors vanishes. Consequently, an adversary attempting to infer metric boundaries from discrete identifier overlaps faces an exponentially ill-conditioned inverse problem: large, continuous coordinate perturbations in high-dimensional space produce negligible or zero changes in exact matches at discrete sub-segment boundaries.

Summary. Solving Voronoi cell intersections requires handling a combinatorial complexity of $O(N^{\lceil D/2 \rceil})$, which is computationally feasible only in low dimensions ($\leq$ 4D) [32]. In 100+-dimensional embedding spaces protected by standard symmetric primitives, continuous coordinate reconstruction from discrete, unranked access patterns is both practically and computationally intractable.

Remark on Schemes Based on Distance-Preserving Encryption. A distinct line of research explores Approximate Distance-Comparison-Preserving Symmetric Encryption (DCPE) or Distance-Recoverable Encryption (DRE) [73], [74]. These schemes enable the untrusted server to directly evaluate pairwise distance inequalities (e.g., determining whether $\|\mathbf{v}_i - \mathbf{q}\| < \|\mathbf{v}_j - \mathbf{q}\|$) over ciphertexts, thereby allowing plaintext graph traversals or clustering algorithms to execute natively on the server.

We emphasize that such schemes operate under a fundamentally weaker security model and are deliberately excluded as private vector search baselines for the following reasons:

- **Severe Metric Leakage and Weak Security Guarantees:** DCPE schemes can at best satisfy relaxed indistinguishability notions (such as IND-DCPA) and are intrinsically vulnerable under passive Known-Plaintext Attacks (KPA). By explicitly exposing the continuous relative rankings or quantized distances between queries and database vectors, they leak the underlying geometric topology to the server.
- **Trivial Susceptibility to Reconstruction Attacks:** As analyzed in §III-A and §A-C, the resilience of EHSearch against coordinate recovery relies critically on the absence of distance and ranking leakage. In sharp contrast, if distance comparisons are revealed (as in DCPE), the continuous coordinates of database embeddings can be reconstructed with high fidelity via non-metric multidimensional scaling, triangulation attacks, or linear programming using only a handful ($O(D)$) of known reference points.

Consequently, while DCPE-based methods can achieve fast retrieval, they do so by trading away semantic vector confidentiality. EHSearch protects the original vectors with IND-CPA security.

APPENDIX B
THEORETICAL PROOF OF HGI COVERAGE AND SEARCH COMPLETENESS

In this section, we formally prove that HGI guarantees both *graph coverage completeness* (no orphan vectors are lost during indexing) and *topological navigability preservation* (the greedy routing paths of HNSW remain reachable within retrieved blocks).

A. Structural Coverage Completeness (Zero Orphan Guarantee)

Let $G_0 = (\mathcal{V}_0, \mathcal{E}_0)$ be the dense Layer-0 HNSW graph, where $|\mathcal{V}_0| = N$, and let $E \subset \mathcal{V}_0$ denote the non-empty set of anchors promoted to upper layers ($|E| = N_{upper} \geq 1$).

Theorem 2 (Indexing Completeness)**.** *For any group capacity limit $B \geq 1$, Algorithm 1 outputs a discrete group set $\mathbf{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_t\}$ such that every node in Layer 0 belongs to at least one group:*

$$\bigcup_{i=1}^{t} \mathcal{G}_i = \mathcal{V}_0. \quad (5)$$

Proof. Let $\mathcal{V}_{\text{BFS}} = \bigcup_{e_i \in E} \mathcal{G}_i$ denote the set of all nodes visited by the capacity-bounded BFS traversals originating from all anchors E. We consider two exhaustive cases:

- **Case 1 ($\mathcal{V}_{\text{BFS}} = \mathcal{V}_0$):** If the union of BFS expansions covers the entire vertex set $\mathcal{V}_0$, every node $v \in \mathcal{V}_0$ is natively included in at least one group $\mathcal{G}_i$. Thus, $\bigcup_{i=1}^{t} \mathcal{G}_i = \mathcal{V}_0$.
- **Case 2 ($\mathcal{V}_{\text{BFS}} \subset \mathcal{V}_0$):** Suppose there exists a non-empty residual set of unvisited nodes $\mathcal{V}_{\text{res}} = \mathcal{V}_0 \setminus \mathcal{V}_{\text{BFS}}$. In the residual absorption step, for every orphan node $u \in \mathcal{V}_{\text{res}}$, Algorithm 1 identifies its topologically closest anchor $e^* = \arg\min_{e_j \in E} \mathrm{dist}_{G_0}(u, e_j)$ and appends u to $\mathcal{G}^*$. Therefore, $\forall u \in \mathcal{V}_{\text{res}}$, $\exists \mathcal{G}^*$ such that $u \in \mathcal{G}^*$.

Combining both cases, $\forall v \in \mathcal{V}_0$, $v \in \bigcup_{i=1}^{t} \mathcal{G}_i$. Hence, HGI provably guarantees 100.00% node coverage, leaving zero unreachable orphan nodes. □

B. Greedy Navigability and Search Completeness

Theorem 3 (Navigability Preservation via Topological Inlining)**.** *Let $P = (e^* = u_0, u_1, \ldots, u_\ell = v^*)$ be the local greedy routing path in Layer 0 from the entry anchor e^* to the target nearest neighbor v^*. If the topological path length ℓ satisfies $\ell \leq \mathrm{depth}(B)$, then all vertices along P are physically colocated within the retrieved block $\mathcal{G}^*$, and local exact reranking is guaranteed to inspect v^*.*

Proof. By construction of HNSW [21], proximity graphs possess the small-world property: the average shortest path length from an upper-layer entry point e^* to any local minimum scales logarithmically with the graph size:

$$\mathbb{E}[\ell] = O\left(\frac{\log N}{\log M}\right), \quad (6)$$

where M is the maximum node degree bound (typically $M = 32$). Because BFS traversal explores nodes strictly ordered by

topological hop distance from e^*, a group capacity of $B \geq 1000$ captures all nodes within a topological radius:

$$\text{depth}(B) \approx \log_M B \approx \frac{\log 1000}{\log 32} \approx 2 \sim 3 \text{ hops.} \quad (7)$$

In small-world routing, the high clustering coefficient ensures that the vast majority of local search paths terminate within this 2- to 3-hop local basin. Furthermore, for boundary targets spanning adjacent clusters, the overlapping boundary padding (§IV-C) ensures that nodes within the intersection of Voronoi boundaries are redundantly colocated in both $\mathcal{G}_i$ and $\mathcal{G}_j$. Because the client retrieves $\tau \geq 20$ target groups in parallel, the probability that the entire greedy chain is covered within $\bigcup_{k \in \mathcal{S}_{\text{target}}} \mathcal{G}_k$ converges to 1, guaranteeing search completeness. □

APPENDIX C
FULL PERFORMANCE ANALYSIS

A. Network Round-Trips

Latency in Wide Area Networks is primarily bounded by network round-trip times. Graph-based schemes, whether secured via ORAM (e.g., Compass) or PIR (e.g., Pacmann), inherently require sequential data accesses to resolve the graph path. Even with batching optimizations in Pacmann, the number of RTTs remains bounded by the hop count $O(L)$. By contrast, EHSearch leverages Topology Distillation to encapsulate the necessary routing neighborhood into spatially contiguous blocks. This effectively decouples the execution flow, reducing network interactions to a single round trip, matching the network efficiency of the IVF-based system.

B. Communication Bandwidth

While Pacmann eliminates the stateful tracking in ORAM, the bandwidth overhead scales sublinearly, $O(\sqrt{N} \cdot d)$ per hop, due to cryptographic primitive replies. Tiptoe fetches the full data for target clusters, consuming $O(\tau \cdot B \cdot d)$ bandwidth. EHSearch minimizes bandwidth consumption by retrieving only a highly refined set of candidates. The uplink payload consists of $O(\tau(m + rD))$ compact trapdoors, while the downlink payload contains exactly $|C|$ encrypted vectors. Because the server-side pigeonhole filtering efficiently prunes false positives, $|C| \ll \tau \cdot B$, thereby minimizing bandwidth consumption.

C. Server Filtering Compute

During query processing, the computational complexity of filtering data and identifying nearest neighbors is a major system bottleneck. In Compass, the server must process ORAM tree paths and handle encrypted payloads of dimension d at each hop, resulting in $O(L \cdot (\log N + d))$ time. Pacmann forces the server to perform matrix multiplications over the entire database to maintain oblivious access, resulting in a filtering computation of $O(L \cdot N \cdot d)$. Tiptoe demands $O(\tau \cdot B \cdot d)$ dense HE multiplications to compute distances. In EHSearch, server-side filtering is transformed into exact table lookups. For a radius r, the query generates $m + rD$ sub-segment trapdoors (or $m + rD$ per target group if group-specific PRF keys are used). Each lookup takes $O(1)$ time, and the server performs $O(\tau(m + rd))$ lookups across τ target groups. Consequently, the filtering compute complexity is $O(\tau(m + rd))$, which reduces to $O(\tau m)$ when $r = 0$. This design ensures that the computational cost of secure filtering is nearly independent of the database size N.

D. Storage Complexity

To achieve this compute-free filtering, EHSearch incurs storage amplification on the server. The server must store the encrypted vectors alongside the inverted multi-index, which requires $O(N_{groups} \cdot B \cdot m)$ entries. This trades server-side storage capacity for minimized computational latency. On the client side, to maintain routing logic without relying on heavy cryptographic state maps (ORAM/PIR), the client stores only the sparse upper layers of the HNSW. This requires $O(N_{upper})$ memory, which typically accounts for less than 3% of the dataset, thereby limiting the client overhead.

APPENDIX D
THEORETICAL MEMORY CONSUMPTION ANALYSIS

As discussed in Section E, scaling to billion-scale datasets introduces memory constraints. In this section, we theoretically analyze the memory footprint of EHSearch during both the **Offline Construction** and the **Online Search** phases as functions of the dataset size N and vector dimensionality D. We denote the HNSW degree bound as M, the group capacity as B, and the number of sub-segments as m. Let N_{upper} be the number of nodes in the HNSW upper layers (Layer ≥ 1), which typically accounts for $1\% \sim 3\%$ of N.

A. Offline Construction Memory

The offline construction phase builds the HGI index and the Group-Scoped Multi-Indexes. Peak memory consumption occurs during HNSW graph construction and subsequent Topology Distillation.

- **Raw Vectors:** The database vectors must be loaded into memory, consuming $N \times D \times 4$ bytes (assuming float32).
- **HNSW Graph Structure:** The adjacency lists of the HNSW graph require memory proportional to the number of edges. In Layer 0, each node can have up to $2M$ neighbors, storing each neighbor ID as a 32-bit integer; therefore, it requires $N \times 2M \times 4$ bytes.
- **RaBitQ Binary Codes:** The quantized D-bit binary strings require $N \times \frac{D}{8}$ bytes.
- **Index Building Buffers:** The BFS traversal and the construction of the multi-index sub-tables require auxiliary buffers, which are bounded by $O(N \cdot m)$.

Therefore, the total peak memory during offline construction is:

$$\mathsf{Mem}_{\text{offline}} = O\big(N \cdot (D + M)\big).$$

Analysis: The memory scales linearly with both N and D. For a billion-scale dataset (e.g., $N = 10^9, D = 128$), the raw vectors alone consume $\approx$ 512 GB, and the graph structure

consumes $\approx$ 256 GB (assuming $M = 32$ and storing $2M$ Layer-0 neighbor IDs per node). This theoretically confirms the necessity of the Memory-Bounded Topology Approximation introduced in Section E.

B. Online Search Memory

The online search phase involves the client and the server operating in a decoupled manner. Their memory footprints differ significantly.

1) Client-Side Memory: The client maintains the local routing cache and processes the query results.

- **Routing Cache:** The client stores the upper layers of the HNSW graph for local routing. This requires storing the vectors and adjacency lists for N_{upper} nodes: $N_{upper} \times (D \times 4 + M \times 8)$ bytes.
- **Query Execution Buffer:** During a query, the client receives the candidate set of size $|C|$, decrypts them, and performs exact reranking. This requires $|C| \times D \times 4$ bytes.

Therefore, the client-side memory is:

$$\mathsf{Mem}_{\text{client}} = O\big(N_{upper} \cdot (D + M) + |C| \cdot D\big).$$

Analysis: Since $N_{upper} \ll N$ (typically $N_{upper} \leq 0.03N$) and $|C|$ is a small constant (typically hundreds to thousands), the reranking buffer remains small. However, the routing cache itself is not negligible at billion scale. For example, with $N = 10^6$, $D = 128$, $M = 32$, and $N_{upper} \approx 0.03N$, the routing cache requires $N_{upper}(4D + 8M) \approx 3 \times 10^4 \times (4 \cdot 128 + 8 \cdot 32) \approx 23$ MB. For $N = 10^9$, a naive plaintext cache of the same upper layers would require about 23 GB. Therefore, billion-scale deployments rely on the memory-bounded topology approximation and compressed routing layer described in Appendix E, rather than caching all upper-layer vectors in plaintext. The reranking buffer, by contrast, still fits comfortably within CPU L1/L2 caches.

2) Server-Side Memory: The server stores the encrypted dataset and the Group-Scoped Multi-Indexes.

- **Encrypted Vectors:** Each vector is encrypted using AES-CBC, introducing a 16-byte IV overhead. The storage is $N \times (D \times 4 + 16)$ bytes.
- **Multi-Index Sub-tables:** For $t = \lceil \frac{N}{B} \rceil$ groups, each containing m sub-tables, the total number of index entries is approximately $N \times m$. Each entry stores an encrypted key and a list of encrypted vector IDs, consuming roughly $O(N \cdot m)$ memory.

Therefore, the server-side memory is:

$$\mathsf{Mem}_{\text{server}} = O\big(N \cdot (D + m)\big).$$

Analysis: The server memory scales linearly with the dataset size. For billion-scale datasets that exceed the RAM capacity of a single machine, the server can offload the $O(N \cdot D)$ encrypted vectors to SSDs, relying on the I/O-Optimized Reranking strategy, while keeping the $O(N \cdot m)$ multi-index sub-tables in RAM for efficient $O(1)$ lookups.

APPENDIX E
SCALING TO BILLION-SCALE DATASETS

To ensure the system remains cost-efficient and practical at this extreme scale, we introduce two necessary architectural compromises: *1) Memory-Bounded Topology Approximation.* A core tenet of HGI is distilling topology from the HNSW upper layers. However, constructing a global HNSW index for billion-scale vectors necessitates an unacceptable memory footprint. To avoid this situation, we introduce a topology approximation strategy: we partially replace the HNSW-derived entry points with K-Means-generated cluster centroids. While K-Means inherently lacks the long-range navigability of HNSW graphs, it provides a memory-efficient, uniformly distributed routing layer. To compensate for the diminished long-range connectivity, we increase the overlapping boundary padding (as described in §IV-C), ensuring that cross-cluster true nearest neighbors remain recoverable. *2) I/O-Optimized Reranking.* For massive datasets, encrypted raw vectors must reside on SSDs rather than in RAM. Executing naive random reads via `mmap` for exact Euclidean reranking triggers severe I/O latency spikes. To address this, we implement a global sorting buffer for the candidate IDs retrieved from the filtering phase. By globally sorting these IDs before issuing `mmap` fetches, we transform fragmented, random 128-byte reads into sequential, page-friendly memory accesses.

APPENDIX F
EXHAUSTIVE PARAMETER SENSITIVITY AND GRID BENCHMARKS

This section reports the comprehensive parameter evaluation on SIFT-1M, GIST-1M, arXiv, and OpenAI. Table V highlights representative operating points along the Pareto frontier across varying sub-segment counts m, bitwidths bps, group capacities B, Hamming radii r, and retrieval depths τ. **Rationale for Default Evaluation Configurations.** In the primary comparative evaluations (Section IX, Table III), we establish default configurations that strike the optimal Pareto trade-off for realistic production serving:

- **Code Granularity (m and bps):** We fix $bps = 16$ and set $m = \frac{D}{16}$ across all datasets ($m = 8$ for SIFT, $m = 48$ for arXiv, $m = 60$ for GIST, and $m = 96$ for OpenAI). This configuration preserves full-dimensional angular resolution ($m \times bps = D$) while ensuring 16-bit word alignment for efficient CPU SIMD vectorization and PRF throughput.
- **Hamming Radius ($r = 1$):** While setting $r = 0$ delivers extreme throughput, its recall is bounded by the strict exact-match requirement. Setting $r = 1$ allows 1-bit flip variations in trapdoors, providing the requisite candidate diversity to surpass the 0.85 recall barrier and enter the high-accuracy regime (Recall@10 ≥ 0.90).
- **Group Capacity (B) and Depth (τ):** We select $B \in \{500, 1000\}$ and $\tau \in \{20, 50\}$ ($\{B = 1000, \tau = 20\}$ for SIFT; $\{B = 500, \tau = 50\}$ for arXiv and OpenAI; $\{B = 1000, \tau = 50\}$ for GIST). Retrieving $\tau = 20 \sim 50$ target groups in a single round reliably covers cross-cluster boundary neighbors to recover the target

TABLE V: Representative parameter configurations across all four benchmark datasets. $|C|$ denotes the average candidate vectors fetched per query. bps denotes bits per segment.

Dataset	Code Bits ($m \times bps$)	Capacity B	Radius r	Groups τ	Candidates $\lvert C\rvert$	QPS	Recall@10
SIFT-1M	120 (10×12)	1000	0	20	6139.9	2470.1	0.8056
	128 (16×8)	1000	0	10	7011.2	2336.0	0.8870
	128 (16×8)	1000	0	20	12195.6	1145.3	0.9467
	128 (8×16)	500	1	20	4735.3	3838.3	0.8417
	128 (8×16)	1000	1	20	8148.2	1650.2	0.9057
	128 (8×16)	1500	1	20	11016.6	1205.0	0.9237
	128 (8×16)	2000	1	20	13534.9	961.3	0.9322
	128 (8×16)	3000	1	20	17910.9	717.3	0.9406
	128 (10×12)	1000	1	50	24696.2	531.0	0.9853
	128 (16×8)	2000	1	50	45589.6	283.3	0.9956
	128 (16×8)	3000	1	50	60131.8	214.0	0.9976
arXiv	768 (48×16)	500	0	20	2373.3	968.2	0.6687
	768 (48×16)	1500	0	20	5693.2	400.4	0.7401
	256 (16×16)	500	1	20	4649.3	508.0	0.7819
	512 (32×16)	500	1	30	8744.4	273.8	0.8554
	768 (48×16)	500	1	20	7063.3	338.0	0.8298
	768 (48×16)	500	1	50	15837.5	147.2	0.9033
	768 (48×16)	1500	1	20	18480.5	130.0	0.9234
	768 (48×16)	1500	1	30	25807.6	93.0	0.9399
	768 (48×16)	1500	1	50	39164.7	58.9	0.9628
	768 (48×16)	1500	1	100	67058.4	16.0	0.9809
GIST-1M	128 (8×16)	500	0	20	2981.3	512.0	0.4111
	960 (60×16)	1000	0	20	12235.2	134.9	0.7534
	256 (16×16)	500	1	20	7520.7	238.8	0.6605
	512 (32×16)	500	1	30	11213.6	159.1	0.7229
	960 (60×16)	500	1	50	17476.9	101.7	0.7923
	960 (60×16)	1000	1	20	14788.1	116.4	0.7622
	960 (60×16)	1000	1	50	30741.3	50.9	0.8752
	960 (60×16)	1000	1	100	51990.8	15.6	0.9278
	960 (60×16)	1000	1	200	84857.3	10.0	0.9612
OpenAI	1536 (96×16)	500	0	20	147.3	5463.7	0.2595
	1536 (96×16)	300	0	20	107.7	6213.9	0.2461
	256 (16×16)	500	1	20	285.5	3397.2	0.3139
	512 (32×16)	500	1	30	738.9	1380.3	0.5038
	1536 (96×16)	300	1	50	1911.0	502.5	0.7724
	1536 (96×16)	500	1	20	1412.8	728.2	0.7563
	1536 (96×16)	500	1	50	2687.7	349.0	0.8068
	1536 (96×16)	500	1	100	4283.4	195.1	0.8339
	1536 (96×16)	500	1	400	10929.4	73.6	0.8596
	1536 (96×16)	500	1	1200	21147.8	35.4	0.8668

Recall@10 ≈ 0.90 (aligned with Compass [11]), while preventing network payload bloat.

For latency-tolerant applications requiring extreme accuracy ceilings ($> 0.96 \sim 0.99$), EHSearch provides a continuous tuning knob by scaling τ (e.g., up to $\tau = 200 \sim 1200$) or expanding B, as demonstrated by the extended frontier points in Table V.