Approximate Vector Set Search: A Bio-Inspired Approach for High-Dimensional Spaces

Yiqi Li[†], Sheng Wang[†*], Zhiyu Chen[‡], Shangfeng Chen[†], and Zhiyong Peng[†§*]
[†]School of Computer Science, Wuhan University
[‡]Amazon.com, Inc. [§]Big Data Institute, Wuhan University
[bruceprofession, swangcs, brucechen, peng]@whu.edu.cn, zhiyuche@amazon.com

***Abstract*—Vector set search, an underexplored similarity search paradigm, aims to find vector sets similar to a query set. This search paradigm leverages the inherent structural alignment between sets and real-world entities to model more fine-grained and consistent relationships for diverse applications. This task, however, faces more severe efficiency challenges than traditional single-vector search due to the combinatorial explosion of pairings in set-to-set comparisons. In this work, we aim to address the efficiency challenges posed by the combinatorial explosion in vector set search, as well as the curse of dimensionality inherited from single-vector search. To tackle these challenges, we present an efficient algorithm for vector set search, `BioVSS` (Bio-inspired Vector Set Search). `BioVSS` simulates the fly olfactory circuit to quantize vectors into sparse binary codes and then designs an index based on the set membership property of the Bloom filter. The quantization and indexing strategy enables `BioVSS` to efficiently perform vector set search by pruning the search space. Experimental results demonstrate over 50 times speedup compared to linear scanning on million-scale datasets while maintaining a high recall rate of up to 98.9%, making it an efficient solution for vector set search.**

I. Introduction

Vector search is a fundamental computational problem in various domains such as information retrieval [18], recommender systems [31], and computer vision [21]. The majority of vector search methods, tailored for querying by a single vector, address the curse of dimensionality [19] and the sparsity issue in high-dimensional vector space [50]. In this paper, we study a novel and challenging problem called *Vector Set Search*. Given a query vector set $\mathbf{Q}$, the objective is to find the top-k most relevant vector sets from a vector set database $\mathbf{D} = \{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_n\}$. This problem transcends the boundaries of single-vector search [12] by incorporating matching and aggregation between vectors within each set. This introduces additional computational complexity, for which there are currently no effective solutions.

Applications. Vector set search has many applications due to its structural alignment with real-world entities, for example:

• *Academic Entity Discovery.* Academic entity discovery facilitates effective exploration of the rapidly growing literature. By representing academic entities as vector sets, such as a paper's profile [48] consisting of vectors representing its citations, or an author's profile [38] as a set of vectors representing their publications, vector set search enables effective identification of relevant research and tracking of academic trends.

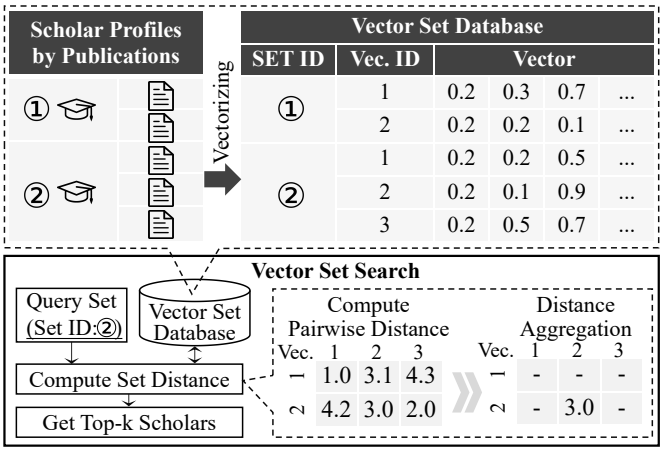


Figure 1: An Exemplar Data Flow of Vector Set Search

• *Multi-Modal Query Answering.* Multi-modal query answering systems leverage vector set search to enhance accuracy. As demonstrated in [43], this approach encodes multi-modal data (e.g., images, text, and audio) into vector sets using a multi-modal encoder [26]. Vector set search then enables accurate queries across multi-modal representations, improving the overall performance of multi-modal query answering systems.

• *Multi-Vector Ranking for Recommender.* Multi-vector ranking enhances recommendation systems by representing users with vector sets. This approach, as demonstrated in [30], represents users as vector sets to capture various aspects of user profiles, including browsing history, search queries, click records, etc. The ranking process compares these vector sets, enabling flexible matching between diverse user interests.

Let us delve into scholar search, an exemplar of vector set search that serves as a key focus in our experimental study.

Example 1: Figure 1 illustrates an exemplar data flow of vector set search applied to scholar search [38], preceded by vector set database preparation. The vector set database construction involves representing scholar profiles as vector sets derived from their publications [9], creating the database of scholars. The vector set search then operates on this database, comprising two key stages: distance computation and neighbor identification. The distance computation stage involves calculating the pairwise distances (such as Euclidean distance) between the query vector set and another set in the

[*]Sheng Wang and Zhiyong Peng are the corresponding authors.
[‡]Work does not relate to position at Amazon.

database, followed by aggregating (such as combining max and min) into set-to-set distances. The neighbor identification phase then determines the top-k nearest scholars based on these set distances for the query. This process enables the discovery of similar scholars for subsequent research activities.

Motivation. The example above reveals a key insight: the structure of vector sets aligns well with real-world entities, enabling effective applications. Recent advancements in embedding models [9] that generate high-dimensional, semantically rich vectors have further expanded the applicability. However, the computational demands of high-dimensional vectors and the combinatorial explosion of pairwise comparisons [3] complicate the development of efficient search.

Recent relevant studies such as `DESSERT` [11] utilize hash tables to accelerate specific non-metric distance computation. However, the fundamental requirements for vector set similarity assessment — precise differentiation capability and consistent measurement across comparisons — demand careful consideration of distance metrics (detailed analysis in Appendix VII-A). The Hausdorff [16], [17] distance naturally aligns with these requirements, providing mathematically rigorous set-to-set similarity measurement while maintaining essential symmetric properties for reliable similarity assessment.

Motivated by these fundamental requirements for robust set comparison, we recognize that metric distances are crucial for consistent evaluations. The Hausdorff distance, a well-studied metric in set theory [16], [17], offers native support for set distance measurement. However, it requires complex pairwise distance calculations, making it computationally intensive. This computational burden underscores the urgent need for more efficient vector set search methods.

Challenges. To provide efficient vector set search in high-dimensional spaces using Hausdorff distance, several key challenges arise. The first challenge is *the curse of dimensionality* [19]. Recent methods address this challenge through hash table construction [11] and the transformation of vector set to single vector [23]. However, both methods are limited to specific non-metric distances, constraining their applicability in Hausdorff distance. The second challenge is the *Aggregation complexity*. Existing Hausdorff distance search algorithms [44], [1], [32] leverage geometric information to decrease the number of aggregation operations. However, geometric information loses effectiveness in high dimensions.

Contributions. To overcome the above two challenges, we propose an algorithm to support an approximate top-k vector set search. Specifically, our work draws inspiration from a locality-sensitive hashing [8], [37] inspired by the olfactory circuit [27]. We simulate the olfactory circuit to quantize vectors into sparse binary codes, addressing the curse of dimensionality. Based on this, an index is designed utilizing the set membership property of Bloom filter, which reduces the search space and decreases the number of aggregation operations. Overall, our contributions are summarized as follows:

- We define the first approximate vector set search problem in high-dimensional spaces using Hausdorff distance, which is a native set metric distance (see Section III).
- We propose `BioVSS`, which uses locality-sensitive property to accelerate vector set search, and provide a comprehensive theoretical analysis with proofs validating the correctness of the proposed method (see Section IV).
- We present an enhanced version of `BioVSS` called `BioVSS++`, which employs a dual-layer cascaded filter composed of inverted index and vector set sketches to reduce unnecessary scans (see Section V).
- We conduct extensive experiments showing our method achieves over 50 times speedup compared to linear scanning on million-scale datasets while maintaining a recall of up to 98.9%, validating its efficiency (see Section VI).

II. RELATED WORK

Single-Vector Search. Single-vector search [49] is increasingly becoming a unified paradigm with the development of embedding models. However, efficiency remains a critical challenge for practical application. Various approximate search algorithms [25] have been developed to perform efficient single-vector search. These algorithms fall into three main categories: locality-sensitive hashing methods [37], [37], [47], [51], graph-based methods [28], and space partition methods [24], [15]. Locality-sensitive hashing like `FlyHash` [37] maps similar vectors to the same hash buckets. Graph-based approaches such as `HNSW` [28] construct navigable graphs for efficient search. Space partitioning methods, such as `IVFFLAT` [49], [10], `IndexIVFPQ` [14], [20], [10] and `IVFScalarQuantizer` [10], utilize `k-means` [46] clustering to partition the vector space and build inverted file indices for fast search. However, existing approaches focus solely on single-vector rather than set-based queries.

Vector Set Search. The problem of vector set search remains relatively unexplored and has not received sufficient attention in the research community. Early research focused on low-dimensional vector space, treating vector set search as a set matching problem often solved with the Kuhn-Munkres algorithm [22]. In geospatial research, trajectories were typically represented as point collections and measured with the Hausdorff distance [2], [44], [45]. Researchers developed various optimizations, such as incorporating road network information [36], using branch-and-bound techniques [32], and employing estimated Hausdorff distances for accelerated computations [1]. However, these approaches, tailored for low-dimensional spaces, do not extend well to high-dimensional spaces. Recent advancements in embedding models enable high-dimensional vectors to carry rich semantic information. By transforming vector sets into single vectors [23], existing high-dimensional single-vector approximate search algorithms can be leveraged to speed up the computational process. However, this transformation relies on a specific distance metric proposed by the work. While the method proposed in [11] is an efficient algorithm that accelerates the vector set search process through hash table construction, it has limitations in terms of measures. Specifically, this approach lacks theoretical support for the Hausdorff distance metric, particularly when

the *max* function is used as the outer aggregation strategy. Moreover, these methods lack effective support for Hausdorff distance. To address this gap, we focus on high-dimensional vector set search using Hausdorff distance.

Bio-Inspired Hashing & Bloom Filter. Olfactory systems in various species show remarkable precision in identifying odorants [6]. For instance, Mice (rodent animals) analyze neural responses in the olfactory bulb to discriminate odors [35]. Similarly, Zebrafish (teleost animals) detect water-dissolved amino acids for olfactory signaling [33]. Furthermore, the olfaction of fly (insect animals) research is the most in-depth and has led to the development of a sparse locality-sensitive hashing function [8], [37] through computational simulations. Fly-inspired hashing [8], a novel locality-sensitive hashing function for single vector search, draws inspiration from the fly olfactory circuit. In the olfactory circuit, only a few neurons respond to specific odor molecules, creating sparse activation patterns that enhance odor recognition efficiency. `BioHash` [37] improves accuracy by learning the intrinsic patterns of data to set the connection strengths of projection neurons in the fly olfactory circuit [42]. Similarly, a Bloom filter maps set elements into a sparse array using hash functions, enabling efficient set storage. The binary Bloom filter [42] includes a binary array, where each position is either 0 or 1. Additionally, the count Bloom filter [5] offers finer granularity through counting. This work leverages the locality-sensitive property of the fly olfactory circuit and the set-storing capabilities of Bloom filters. Meanwhile, we exploit their structural similarities to construct an efficient index.

III. DEFINITIONS & PRELIMINARIES

Definition 1 (**Vector**). *A **vector** $\mathbf{v} = (v_1, v_2, ..., v_d)$ is a tuple of real numbers, in which $v_i \in \mathbb{R}$ and $d \in \mathbb{N}^+$ represents the dimensionality of the vector.*

Definition 2 (**Vector Set**). *A **vector set** $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m\}$ contains a set of vectors, where $m \in \mathbb{N}^+$ is number of vectors.*

Definition 3 (**Vector Set Database**). *A **vector set database** $\mathbf{D} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$ is a collection of vector sets, where each $\mathbf{V}_i$ is a vector set and $n \in \mathbb{N}^+$ is the number of vector sets.*

Definition 4 (**Hausdorff Distance**). *Given two vector sets $\mathbf{Q}$ and $\mathbf{V}$, the **Hausdorff distance** from $\mathbf{Q}$ to $\mathbf{V}$ is defined as:*

$$Haus(\mathbf{Q}, \mathbf{V}) = \max\left(\max_{\mathbf{q}\in\mathbf{Q}} \min_{\mathbf{v}\in\mathbf{V}} dist(\mathbf{q}, \mathbf{v}), \max_{\mathbf{v}\in\mathbf{V}} \min_{\mathbf{q}\in\mathbf{Q}} dist(\mathbf{v}, \mathbf{q})\right),$$

where $dist(\mathbf{q}, \mathbf{v}) = \|\mathbf{q} - \mathbf{v}\|_2$ is the Euclidean distance between vectors $\mathbf{q} \in \mathbb{R}^d$ and $\mathbf{v} \in \mathbb{R}^d$.

To elucidate the computational process of Hausdorff distance, let us consider a concrete example.

Example 2: The computation process of Hausdorff distance is illustrated in Figure 2. Given two finite vector sets $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$ and $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2\}$ in a high-dimensional spaces, the Hausdorff distance $Haus(\mathbf{Q}, \mathbf{V})$ is calculated through the following steps: 1) calculate the maximum of the minimum distances from each vector in $\mathbf{V}$ to all vectors in $\mathbf{Q}$, denoted as $\max_{\mathbf{v}\in\mathbf{V}} \min_{\mathbf{q}\in\mathbf{Q}} d(\mathbf{v}, \mathbf{q}) = 2$; 2) calculate the maximum of the minimum distances from each vector in $\mathbf{Q}$ to all vectors in $\mathbf{V}$, expressed as $\max_{\mathbf{q}\in\mathbf{Q}} \min_{\mathbf{v}\in\mathbf{V}} d(\mathbf{q}, \mathbf{v}) = 3$; 3) determine the Hausdorff distance by max aggregating the results from steps 1 and 2 to obtain $Haus(\mathbf{Q}, \mathbf{V}) = 3$.

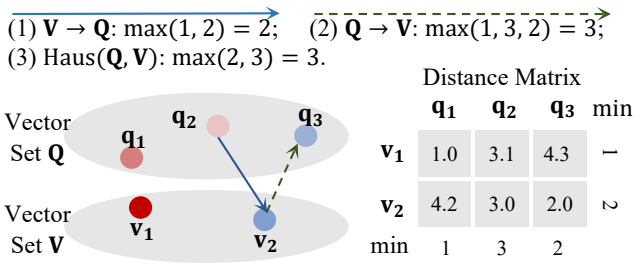


Figure 2: The Calculation Process of Hausdorff Distance

Hausdorff distance calculation involves pairwise distance computations between vectors from two sets, followed by aggregation as described in Definition 4. The computational complexity for the Hausdorff distance is $O(m^2 \cdot d)$, where m is the number of vectors per set and d is the vector dimensionality. This quadratic dependence on m makes exact computation prohibitive for large-scale datasets, especially as set sizes and dimensionality d increase.

To reduce computational overhead, extensive single-vector search works have validated the efficacy of approximation techniques such as quantization [8], [37], [47] and index pruning [24], [15], [28], [14], [20], [10]. Inspired by these, we introduce the approximate top-k vector set search problem.

Definition 5 (**Approximate Top-k Vector Set Search**). *Given a vector set database $\mathbf{D} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$, a query vector set $\mathbf{Q}$ and a Hausdorff distance function $Haus(\mathbf{Q}, \mathbf{V})$, the **approximate top-k vector set search** is the task of returning $\mathbf{R}$ with probability at least $1 - \delta$:*

$$\mathbf{R} = \{\mathbf{V}_1^\star, \mathbf{V}_2^\star, ..., \mathbf{V}_k^\star\} = \text{argmin}^k_{\mathbf{V}_i^\star \in \mathbf{D}} Haus(\mathbf{Q}, \mathbf{V}_i^\star),$$

where $\text{argmin}^k_{\mathbf{V}_i \in \mathbf{D}}$ selects k sets $\mathbf{V}_i^\star$ minimizing $Haus(\mathbf{Q}, \mathbf{V}_i^\star)$ and $\mathbf{V}_k^\star$ has the k-th smallest distance from $\mathbf{Q}$. The failure probability is denoted by $\delta \in [0, 1]$.

Approximate top-k vector set search trades off speed and accuracy, permitting a small error in exchange for substantially improved search efficiency.

IV. PROPOSED BIOVSS

We propose `BioVSS`, an efficient search algorithm designed to address the approximate top-k vector set search problem. Drawing inspiration from the fly olfactory circuit, `BioVSS` leverages the locality-sensitive property of the olfactory circuit to enhance search efficiency. Section IV-B establishes theoretical foundations for locality-sensitive Hausdorff distance.

A. Overview Algorithm

`BioVSS` algorithm comprises two principal components: 1) the binary hash encoding for vector sets, and 2) the search execution in `BioVSS`.

TABLE I: Summary of Major Notations

Notation	Description
$\mathbf{D}$	Vector Set Database
$\mathbf{D^H}$	Sparse Binary Codes of $\mathbf{D}$
$\mathbf{T}, \mathbf{Q}$	Vector Set of Target and Query
n, m, m_q	Cardinality of $\mathbf{D}$, $\mathbf{T}$, and $\mathbf{Q}$
$L_{wta} = L$	Number of Hash Functions / Winner-Takes-All
$\mathcal{H}$	Hash Function (see Definition 7)
k	The Number of Results Returned
$\sigma(\cdot)$	Min-Max Similarity Function (see Lemma 1)
$S^{\alpha}_{ij}, S^{\beta}_{ij}, \mathbf{S}$	(i,j)-th Entry of the Similarity Matrix $\mathbf{S}$
$s_{\max}, s_{\min}$	Max. and Min. Real Similarities btw. Vector Sets
$\hat{\mathbf{S}}, \hat{s}_{\max}, \hat{s}_{\min}$	Estimated Values of $\mathbf{S}$, $s_{\max}$, and $s_{\min}$
δ	Vector Set Search Failure Probability
$B^{\star}_{\alpha}, B_{\beta}, B', B^{\star}$	Similarity Bounds of Vector Sets (see Theorem 4)

1) ***Binary Hash Encoding for Vector Set****:* the rationale behind binary hash encoding is rooted in exploiting the inherent similarity-preserving property of hashing to mitigate computational complexity and enhance search efficiency.

First, we define the unified paradigm of `LSH` shared by various `LSH` [8], [37], [47], [51] and proposed by [19].

Definition 6 (**Locality-Sensitive Hashing Function** [19])**.** *An `LSH` hash function $h : \mathbb{R}^d \to \mathbb{R}$ is called a similarity-preserving hash for vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ if*

$$\mathbb{P}(h(\mathbf{a}) = h(\mathbf{b})) = \text{sim}(\mathbf{a}, \mathbf{b}), \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d,$$

where $\text{sim}(\mathbf{a}, \mathbf{b}) \in [0, 1]$ is the similarity between vectors $\mathbf{q} \in \mathbb{R}^d$ and $\mathbf{v} \in \mathbb{R}^d$.

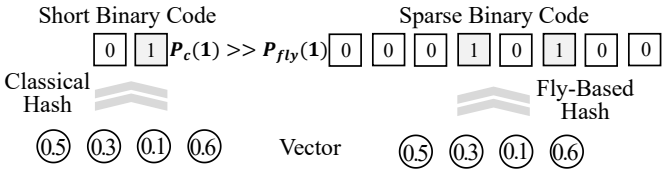
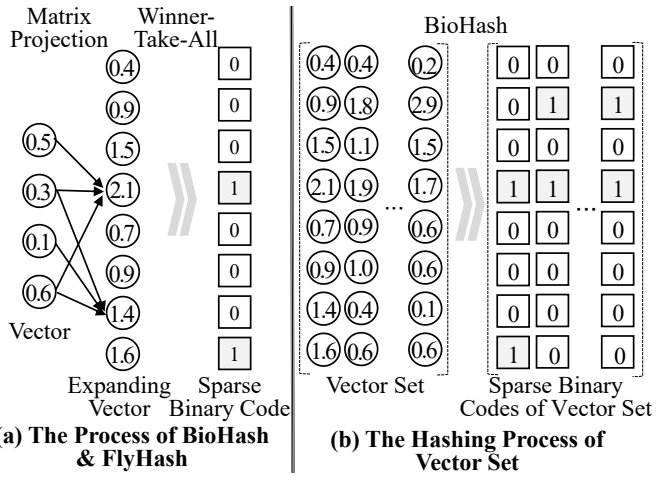


Figure 3: Classical Hashing vs. Fly-Based Hashing

Building on this unified paradigm, we adopt a fly-based hashing approach for our framework. As illustrated in Figure 3, this approach fundamentally differs from classical hashing methods in its dimensional transformation strategy. Classical hashing techniques typically perform dimensionality reduction to generate dense hash codes. In contrast, fly-inspired hashing emulates the neural projection patterns in Drosophila's olfactory circuit [8]. This fly-based approach performs expansion by projections, resulting in sparse binary codes.

The choice of fly-based hashing is motivated by two key factors. First, its sparse binary coding aligns naturally with the set-based structure of Bloom filters, which is essential for our framework's effectiveness (see Section V). This structural compatibility makes it more suitable than traditional distance-preserving hash functions. Second, among fly-based methods, we specifically adopt `BioHash` [37] over the original `FlyHash` [8]. Empirical studies have shown that `BioHash` achieves approximately twice the similarity preservation performance of `FlyHash` [37].

Definition 7 (**FlyHash** [8] & **BioHash** [37])**.** *A `FlyHash` or `BioHash` is a function $\mathcal{H} : \mathbb{R}^d \to \{0, 1\}^b$ that maps a vector $\mathbf{v} \in \mathbb{R}^d$ to a sparse binary vector $\mathbf{h} = \mathcal{H}(\mathbf{v}) \in \{0, 1\}^b$. The hash code is generated as $\mathbf{h} = WTA(W\mathbf{v})$, where $W \in \mathbb{R}^{b \times d}$ is a random projection matrix, and WTA is the Winner-Take-All (`WTA`) operation that sets the L_{wta} largest elements to 1 and the rest to 0. The resulting binary vector $\mathbf{h}$ has L_{wta} non-zero elements ($\|\mathbf{h}\|_1 = L_{wta} \ll b$), where L_{wta} also represents the number of hash functions composing $\mathcal{H}$ defined in Lemma 6.*



(a) The Process of BioHash & FlyHash **(b) The Hashing Process of Vector Set**

Figure 4: The Hashing Process of `BioVSS`

As illustrated in Figure 4(a), `FlyHash` [8] generates `LSH` codes by mimicking the olfactory neural circuit of the fly. Vectors are projected into higher-dimensional spaces using a matrix that simulates projection neurons, enhancing representational capacity. Winner-take-all generates sparse binary codes by selecting the top responding neurons. This has been shown to exhibit locality-sensitive property [8] (Definition 6).

Section IV-B establishes the theoretical foundations of this method. It demonstrates that the hash codes effectively support the approximate top-k vector set search. As illustrated in Figure 4(b), the algorithm quantizes the vector sets into binary hash codes. Next, we introduce the hashing process.

Algorithm 1 outlines the process of generating sparse binary codes for a database of vector sets. The input comprises a database $\mathbf{D}$ of n vector sets $\{\mathbf{V}_i\}_{i=1}^n$ and a parameter L_{wta} for the winner-takes-all operation, while the output $\mathbf{D^H}$ is a collection of corresponding sparse binary codes. The algorithm iterates through each vector set $\mathbf{V}_j$ (line 2), applying the `BioHash` (see Definition 7) to each vector $\mathbf{v}$ (lines 5-6). This involves a matrix projection $W\mathbf{v}$ followed by the `WTA` operation, which sets the L_{wta} largest elements to 1 and the rest to 0, resulting in a binary code $\mathbf{h}$ with exactly L_{wta} non-

Algorithm 1: `Gen_Binary_Codes`($\mathbf{D}, L_{wta}$)

Input: $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^n$: vector set database, L_{wta}: # of `WTA`.
Output: $\mathbf{D^H}$: sparse binary codes.

1 **Initialization:** $\mathbf{D^H} = \emptyset$
2 **for** $j = 1$ **to** n **do**
3 $\mathbf{H}_j = \emptyset$;
4 **for** *each* $\mathbf{v} \in \mathbf{V}_j$ **do**
5 $\mathbf{h}_p = W\mathbf{v}$; `// Matrix projection`
6 $\mathbf{h} = WTA(\mathbf{h}_p, L_{wta})$; `// Winner-takes-all`
7 $\mathbf{H}_j = \mathbf{H}_j \cup \{\mathbf{h}\}$;
8 $\mathbf{D^H} = \mathbf{D^H} \cup \{\mathbf{H}_j\}$;
9 **return** $\mathbf{D^H}$;

Algorithm 2: `BioVSS_Topk_Search`$(\mathbf{Q}, k, \mathbf{D}, \mathbf{D^H}, L_{wta}, c)$

Input: $\mathbf{Q}$: query vector set, k: # of top sets, $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^{n}$: vector set database, $\mathbf{D^H}$: sparse binary codes, L_{wta}: # of `WTA`, c: the size of candidate set.
Output: $\mathbf{R}$: top-k vector sets.

1 **Initialization:** $\mathbf{Q^H} = \emptyset$, $C = \emptyset$;
2 **for** *each* $\mathbf{q} \in \mathbf{Q}$ **do**
3 $\quad \mathbf{h}_p = W\mathbf{q}$; `// Matrix projection`
4 $\quad \mathbf{h}_q = WTA(\mathbf{h}_p, L_{wta})$; `// Winner-takes-all`
5 $\quad \mathbf{Q^H} = \mathbf{Q^H} \cup \{\mathbf{h}_q\}$;
`// Select candidates`
6 **for** $j = 1$ **to** n **do**
7 $\quad d_H = Haus^H(\mathbf{Q^H}, \mathbf{H}_j)$;
8 $\quad C = C \cup \{(\mathbf{V}_j, d_H)\}$;
9 $\mathcal{F} = \{(\mathbf{V}_i, d_H) \in C \mid d_H \leq d_H^{(c)}\}$, where $d_H^{(c)}$ is the c-th smallest d_H in C;
`// Select top-k results`
10 $\mathcal{D} = \emptyset$;
11 **for** *each* $(\mathbf{V}_i, d_H) \in \mathcal{F}$ **do**
12 $\quad d_i = Haus(\mathbf{Q}, \mathbf{V}_i)$;
13 $\quad \mathcal{D} = \mathcal{D} \cup \{(\mathbf{V}_i, d_i)\}$;
14 $\mathbf{R} = \{(\mathbf{V}_i, d_i) \in \mathcal{D} \mid d_i \leq d_i^{(k)}\}$, where $d_i^{(k)}$ is the k-th smallest d_i in $\mathcal{D}$;
15 **return** $\mathbf{R}$;

zero elements. These sparse binary codes are aggregated per vector set (line 7) and compiled into $\mathbf{D^H}$ (line 8).

*2) **Search Execution in `BioVSS`:*** binary codes offer computational advantages by enabling fast bitwise operations, which are inherently supported by modern CPU architectures [29]. This leads to significant speedups compared to traditional floating-point computations.

Algorithm 2 outlines `BioVSS` top-k search process in vector set databases. The input includes a query set $\mathbf{Q}$, the number of desired results k, the database $\mathbf{D}$, pre-computed sparse binary codes $\mathbf{D^H}$, and the `WTA` parameter L_{wta}. The output $\mathbf{R}$ contains the top-k most similar vector sets. The algorithm begins by generating sparse binary codes for the query set $\mathbf{Q}$ (lines 2-5). This creates the binary representation of the query set $\mathbf{Q^H}$. To leverage CPU-friendly bit operations, we compute the Hamming-based Hausdorff distance $Haus^H$ (substituting *dist* with $hamming(\mathbf{q}, \mathbf{v})$ [34] in Definition 4) between $\mathbf{Q^H}$ and each set of binary codes $\mathbf{H}_j$ in $\mathbf{D^H}$ (lines 6-8). This step efficiently identifies a set of candidate vector sets $\mathcal{F}$ based on their binary code similarities (line 9). The final stage refines these candidates by computing the actual Hausdorff distance (see Definition 4) between $\mathbf{Q}$ and each candidate vector set $\mathbf{V}_i$ in the original space (lines 10-13). The algorithm then selects the top-k results based on these distances (line 14) and then returns $\mathbf{R}$ (line 15).

B. Theoretical Analysis of Algorithm Correctness

This section presents a theoretical analysis of the probabilistic guarantees for result correctness in `BioVSS`. We define a function constraining similarity measure bounds, followed by upper and lower tail probability bounds for similarity comparisons. Through several lemmas, we construct our theoretical framework, culminating in Theorem 4, which establishes the relationship between the error rate δ and $L = L_{wta}$.

Assumptions. Our framework assumes all vectors undergo $L2$ normalization. This allows us to define similarity between vectors $\mathbf{q}$ and $\mathbf{v}$ as their inner product: $sim(\mathbf{q}, \mathbf{v}) = \mathbf{q}^T\mathbf{v}$. We denote the query vector set and another vector set as $\mathbf{Q}$ and $\mathbf{V}$, respectively. To facilitate proof, we define $Sim_{Haus}(\mathbf{Q}, \mathbf{V}) = \min(\min_{\mathbf{q}\in\mathbf{Q}} \max_{\mathbf{v}\in\mathbf{V}} sim(\mathbf{q}, \mathbf{v}), \min_{\mathbf{v}\in\mathbf{V}} \max_{\mathbf{q}\in\mathbf{Q}} sim(\mathbf{q}, \mathbf{v}))$. The similarity metric is equivalent to the Hausdorff distance for $L2$-normalized vectors.

1) Bounds on Min-Max Similarity Scores: we begin by introducing upper and lower bounds for a function σ, which forms the foundation for our subsequent proofs.

Lemma 1. *Consider a matrix $\mathbf{S} = [s_{ij}] \in \mathbb{R}^{m_q \times m}$ containing similarity scores between a query vector set $\mathbf{Q}$ and a target vector set $\mathbf{T}$, where $|\mathbf{Q}| = m_q$ and $|\mathbf{T}| = m$. Define a function $\sigma : \mathbb{R}^{m_q \times m} \to \mathbb{R}$ as $\sigma(\mathbf{S}) = \min(\min_i \max_j s_{ij}, \min_j \max_i s_{ij})$ Then, $\sigma(\mathbf{S})$ satisfies the following bounds: $\min_{i,j} s_{ij} \leq \sigma(\mathbf{S}) \leq \max_{i,j} s_{ij}$.*

Proof. Let $a = \min_i \max_j s_{ij}$ and $b = \min_j \max_i s_{ij}$. Then $\sigma(\mathbf{S}) = \min(a, b)$.

For the lower bound: $\forall i, j : s_{ij} \leq \max_j s_{ij} \Rightarrow \min_{i,j} s_{ij} \leq \min_i \max_j s_{ij} = a$. Similarly, $\min_{i,j} s_{ij} \leq b$. Therefore, $\min_{i,j} s_{ij} \leq \min(a, b) = \sigma(\mathbf{S})$.

For the upper bound: $\forall i : \max_j s_{ij} \leq \max_{i,j} s_{ij} \Rightarrow a = \min_i \max_j s_{ij} \leq \max_{i,j} s_{ij}$. Similarly, $b \leq \max_{i,j} s_{ij}$. Therefore, $\sigma(\mathbf{S}) = \min(a, b) \leq \max_{i,j} s_{ij}$.

Thus, we have $\min_{i,j} s_{ij} \leq \sigma(\mathbf{S}) \leq \max_{i,j} s_{ij}$. □

2) Upper Tail Probability Bound: we now establish the upper tail probability bound for our relevance score function, which forms the foundation for our subsequent proofs.

Lemma 2. *Consider a similarity matrix $\mathbf{S} \in \mathbb{R}^{m_q \times m}$ between a query vector set and a target vector set. Define the maximum similarity score in $\mathbf{S}$ as $s_{\max} = \max_{i,j} s_{ij}$. Given an estimated similarity matrix $\hat{\mathbf{S}}$ of $\mathbf{S}$ and a threshold $\tau_1 \in (s_{\max}, 1)$, we define $\Delta_1 = \tau_1 - s_{\max}$. Then, the following inequality holds:*

$$\Pr\left[\sigma(\hat{\mathbf{S}}) \geq s_{\max} + \Delta_1\right] \leq m_q m \gamma^L,$$

for $\gamma = \left(\frac{s_{\max}(1-\tau_1)}{\tau_1(1-s_{\max})}\right)^{\tau_1} \left(\frac{1-s_{\max}}{1-\tau_1}\right)$. Here, $\sigma(\cdot)$ is the operator defined in Lemma 1, and $L \in \mathbb{Z}^+$ represents the number of hash functions.

Proof. Applying a generic Chernoff bound to $\sigma(\hat{\mathbf{S}})$ yields the following bounds for any $t > 0$:

$$\Pr[\sigma(\hat{\mathbf{S}}) \geq \tau_1] = \Pr\left[e^{t\sigma(\hat{\mathbf{S}})} \geq e^{t\tau_1}\right] \leq \frac{\mathbb{E}\left[e^{t\sigma(\hat{\mathbf{S}})}\right]}{e^{t\tau_1}}.$$

Given an `LSH` family (see Definition 7), the estimated maximum similarity $\hat{s}_{\max}$ follows a scaled binomial distribution with parameters $s_{\max}$ and L^{-1}, i.e., $\hat{s}_{\max} \sim L^{-1}\mathcal{B}(s_{\max}, L)$.

Substituting the binomial moment generating function into the expression and using Lemma 1, we can bound the numerator:

$$\mathbb{E}\left[e^{t\sigma(\hat{\mathbf{S}})}\right] \le \mathbb{E}\left[e^{t\max_{1\le i\le m_q,1\le j\le m}\hat{S}_{ij}}\right] \le m_q m\mathbb{E}\left[e^{t\hat{s}_{\max}}\right] = m_q m\left(1-s_{\max}+s_{\max}e^{\frac{t}{L}}\right)^L.$$

Combining the Chernoff bound and the numerator bound yields: $\Pr[\sigma(\hat{\mathbf{S}})\ge\tau_1] \le m_q m e^{-t\tau_1}\left(1-s_{\max}+s_{\max}e^{\frac{t}{L}}\right)^L$.

To find the tightest upper tail probability bound, we can determine the infimum by setting the derivative of the bound for t equal to zero. Then, we get: $t^\star = L\ln\left(\frac{\tau_1(1-s_{\max})}{s_{\max}(1-\tau_1)}\right)$.

Since $\tau_1 \in (s_{\max}, 1)$, the numerator of the fraction inside the logarithm is greater than the denominator, ensuring that $t^\star > 0$. This result allows us to obtain the tightest bound.

Substituting $t = t^\star$ into the bound, we obtain:

$$\Pr[\sigma(\hat{\mathbf{s}})\ge\tau_1] \le m_q m\left(\left(\frac{\tau_1(1-s_{\max})}{s_{\max}(1-\tau_1)}\right)^{-\tau_1}\left(\frac{1-s_{\max}}{1-\tau_1}\right)\right)^L.$$

Thus we have: $\gamma = \left(\frac{s_{\max}(1-\tau_1)}{\tau_1(1-s_{\max})}\right)^{\tau_1}\left(\frac{1-s_{\max}}{1-\tau_1}\right)$. □

3) Lower Tail Probability Bound: we now establish the lower tail probability bound for our relevance score function, which forms the foundation for our subsequent proofs.

Lemma 3. *Consider a similarity matrix $\mathbf{S}\in\mathbb{R}^{m_q\times m}$ between a query vector set and a target vector set. Define the minimum similarity score in $\mathbf{S}$ as $s_{\min}=\min_{i,j}s_{ij}$. Given an estimated similarity matrix $\hat{\mathbf{S}}$ of $\mathbf{S}$ and a threshold $\tau_2\in(0,s_{\min})$, we define $\Delta_2 = s_{\min}-\tau_2$. Then, the following inequality holds:*

$$\Pr\left[\sigma(\hat{\mathbf{s}}) \le s_{\min}-\Delta_2\right] \le m_q m\xi^L,$$

for $\gamma = \left(\frac{s_{\min}(1-\tau_2)}{\tau_2(1-s_{\min})}\right)^{\tau_2}\left(\frac{1-s_{\min}}{1-\tau_2}\right)$. Here, $\sigma(\cdot)$ is the operator defined in Lemma 1, and $L\in\mathbb{Z}^+$ represents the number of hash functions.

Proof. Applying a generic Chernoff bound to $\sigma(\hat{\mathbf{S}})$ yields the following inequality for any $t<0$:

$$\Pr[\sigma(\hat{\mathbf{S}})\le\tau_2] = \Pr\left[e^{t\sigma(\hat{\mathbf{S}})}\ge e^{t\tau_2}\right] \le \frac{\mathbb{E}\left[e^{t\sigma(\hat{\mathbf{S}})}\right]}{e^{t\tau_2}}.$$

Given an LSH family (see Definition 7), the estimated minimum similarity $\hat{s}_{\min}$ follows a scaled binomial distribution with parameters $s_{\min}$ and L^{-1}, i.e., $\hat{s}_{\min}\sim L^{-1}\mathcal{B}(s_{\min},L)$. Substituting the binomial moment generating function into the expression and using Lemma 1, we can bound the numerator:

$$\mathbb{E}\left[e^{t\sigma(\hat{\mathbf{S}})}\right] \le \mathbb{E}\left[e^{t\min_{1\le i\le m_q,1\le j\le m}\hat{S}_{ij}}\right] \le m_q m\mathbb{E}\left[e^{t\hat{s}_{\min}}\right] = m_q m\left(1-s_{\min}+s_{\min}e^{\frac{t}{L}}\right)^L.$$

Combining the Chernoff bound and the numerator bound yields: $\Pr[\sigma(\hat{\mathbf{S}})\le\tau_2] \le m_q m e^{-t\tau_2}\left(1-s_{\min}+s_{\min}e^{\frac{t}{L}}\right)^L$.

To find the tightest upper bound, we can determine the infimum by setting the derivative of the upper bound with respect to t equal to zero. Then, we obtain: $t^\star = L\ln\left(\frac{\tau_2(1-s_{\min})}{s_{\min}(1-\tau_2)}\right)$.

Since $\tau_2\in(0,s_{\min})$, the numerator of the fraction inside the logarithm is greater than the denominator, ensuring that $t^\star<0$. This result allows us to obtain the tightest upper bound.

Similar to Lemma 2, we get: $\xi = \left(\frac{s_{\min}(1-\tau_2)}{\tau_2(1-s_{\min})}\right)^{\tau_2}\left(\frac{1-s_{\min}}{1-\tau_2}\right)$. □

4) Probabilistic Correctness for Search Results: this section presents a theorem establishing the probabilistic guarantees to the approximate top-k vector set search problem.

Theorem 4. *Let $\mathbf{V}_\alpha^\star$ be one of the top-k vector sets that minimize the Hausdorff distance $Haus(\mathbf{Q},\mathbf{V})$, i.e., $\mathbf{V}_\alpha^\star\in\operatorname{argmin}^k_{\mathbf{V}\in\mathbf{D}}Haus(\mathbf{Q},\mathbf{V})$, and $\mathbf{V}_\beta$ be any other vector set in the database $\mathbf{D}$. By applying Lemma 1, we obtain the following upper and lower bounds for $Haus(\mathbf{Q},\mathbf{V}_\alpha^\star)$ and $Haus(\mathbf{Q},\mathbf{V}_\beta)$:*

$$B_\alpha^\star = 1-\min_{i,j}S_{ij}^\alpha = 1-s_{\alpha,\min},\quad B_\beta = 1-\max_{i,j}S_{ij}^\beta = 1-s_{\beta,\max},$$

where S_{ij}^α and S_{ij}^β denote the element-wise similarity matrices between each vector in the query set $\mathbf{Q}$ and each vector in $\mathbf{V}_\alpha^\star$ and $\mathbf{V}_\beta$.

Let $B' = \min_\beta B_\beta$ be the minimum value of B_β over any vector set $\mathbf{V}_\beta$ not in the top-k results, and let $B^\star = \max_\alpha B_\alpha^\star$ be the maximum value of $B_\alpha^\star$ over any top-k vector set $\mathbf{V}_\alpha^\star$. Define $\Delta_1+\Delta_2$ as follows:

$$B_\beta - B_\alpha^\star = s_{\alpha,\min}-s_{\beta,\max} \ge B'-B^\star = 2(\Delta_1+\Delta_2).$$

If $\Delta_1>0$ and $\Delta_2>0$, a hash structure with $L = O\left(\log\left(\frac{nm_qm}{\delta}\right)\right)$ solves the approximate top-k vector set search problem (Definition 5) with probability at least $1-\delta$, where $n=|\mathbf{D}|$, $m_q=|\mathbf{Q}|$, $m=|\mathbf{V}_i|$ (Assume m is invariant across all $\mathbf{V}_i$), and δ is the failure probability.

Proof. We aim to determine the value of L that satisfies both tail bounds and combine them to obtain the final L.

Upper Tail Probability Bound. For $\mathbf{V}_\beta\notin\operatorname{argmin}^k_{\mathbf{V}\in\mathbf{D}}Haus(\mathbf{Q},\mathbf{V})$, Lemma 2 yields:

$$\Pr\left[\sigma(\hat{\mathbf{S}}_\beta)\ge s_{\beta,\max}+\Delta_1\right] \le m_q m\gamma_\beta^L,$$

where $\gamma_\beta = \left(\frac{s_{\beta,\max}(1-\tau_1)}{\tau_1(1-s_{\beta,\max})}\right)^{\tau_1}\left(\frac{1-s_{\beta,\max}}{1-\tau_1}\right)$.

To simplify the analysis, we consider $\gamma_{\max}=\max_{\mathbf{V}_\beta}\gamma_\beta$, allowing us to express all bounds using the same $\gamma_{\max}$. To ensure that the probability of the bound holding for all $n-k$ non-top-k sets is $\frac{\delta}{2}$, we choose L such that $L\ge\log\frac{2(n-k)m_qm}{\delta}\left(\log\frac{1}{\gamma_{\max}}\right)^{-1}$. Consequently,

$$\Pr\left[\sigma(\hat{\mathbf{S}}_\beta)\ge s_{\beta,\max}+\Delta_1\right] \le m_q m\gamma_\beta^L \le m_q m\gamma_{\max}^L \le m_q m\gamma_{\max}^{\log\frac{2(n-k)m_qm}{\delta}\left(\log\frac{1}{\gamma_{\max}}\right)^{-1}} = \frac{\delta}{2(n-k)}.$$

Lower Tail Probability Bound. For $\mathbf{V}_\alpha^\star\in\operatorname{argmin}^k_{\mathbf{V}\in\mathbf{D}}Haus(\mathbf{Q},\mathbf{V})$, Lemma 3 yields:

$$\Pr\left[\sigma(\hat{\mathbf{S}}_\alpha)\le s_{\alpha,\min}-\Delta_2\right] \le m_q m\xi_\alpha^L,$$

where $\xi_\alpha = \left(\frac{s_{\alpha,\min}(1-\tau_2)}{\tau_2(1-s_{\alpha,\min})}\right)^{\tau_2}\left(\frac{1-s_{\alpha,\min}}{1-\tau_2}\right)$.

Following a similar approach as for the upper bound, we ensure that the probability of the bound holding for all top-k sets is $\frac{\delta}{2}$ by selecting L such that $L \geq \log\frac{2km_qm}{\delta}\left(\log\frac{1}{\xi_{\max}}\right)^{-1}$, where $\xi_{\max} = \max_\alpha \xi_\alpha$. As a result,

$$\begin{aligned}\Pr\left[\sigma(\hat{\mathbf{S}}_\alpha) \leq s_{\alpha,\min} - \Delta_2\right] &\leq m_q m \xi_\alpha^L \leq m_q m \xi_{\max}^L \\ &\leq m_q m \xi_{\max}^{\log\frac{2km_qm}{\delta}\left(\log\frac{1}{\xi_{\max}}\right)^{-1}} \\ &= \frac{\delta}{2k}.\end{aligned}$$

Combining the Bounds. Let $L = \max\left(\frac{\log\frac{2(n-k)m_qm}{\delta}}{\log\left(\frac{1}{\gamma_{\max}}\right)}, \frac{\log\frac{2km_qm}{\delta}}{\log\left(\frac{1}{\xi_{\max}}\right)}\right)$, we calculate the final L by combining the upper and lower tail probability bounds. Let $\mathbb{1}$ be an indicator random variable that equals 1 when all the upper and lower tail probability bounds are satisfied, and 0 otherwise. The probability of solving the approximate top-k vector set problem is equivalent to the probability that:

$$\begin{aligned}
&\Pr\left(\forall_{\alpha,\beta}\left(\hat{H}\left(\mathbf{Q},\mathbf{V}_\alpha^\star\right) - \hat{H}\left(\mathbf{Q},\mathbf{V}_\beta\right) < 0\right)\right) \\
&= \Pr\left(\forall_{\alpha,\beta}\left(\min\left(\min_{\mathbf{q}\in\mathbf{Q}}\max_{\mathbf{v}\in\mathbf{V}_\alpha^\star}(sim(\mathbf{q},\mathbf{v})), \min_{\mathbf{v}\in\mathbf{V}_\alpha^\star}\max_{\mathbf{q}\in\mathbf{Q}}(sim(\mathbf{q},\mathbf{v}))\right)\right.\right. \\
&\quad \left.\left. - \min\left(\min_{\mathbf{q}\in\mathbf{Q}}\max_{\mathbf{v}\in\mathbf{V}_\beta}(sim(\mathbf{q},\mathbf{v})), \min_{\mathbf{v}\in\mathbf{V}_\beta}\max_{\mathbf{q}\in\mathbf{Q}}(sim(\mathbf{q},\mathbf{v}))\right) > 0\right)\right) \\
&= \Pr\left(\forall_{\alpha,\beta}\left(\min\left(\min_{1\leq i\leq m_q}\max_{1\leq j\leq m} S_{ij}^\alpha, \min_{1\leq j\leq m}\max_{1\leq i\leq m_q} S_{ij}^\alpha\right)\right.\right. \\
&\quad \left.\left. - \min\left(\min_{1\leq i\leq m_q}\max_{1\leq j\leq m} S_{ij}^\beta, \min_{1\leq j\leq m}\max_{1\leq i\leq m_q} S_{ij}^\beta\right) > 0\right)\right) \\
&\geq \Pr\left(\forall_{\alpha,\beta}\left(\min\left(\min_{1\leq i\leq m_q}\max_{1\leq j\leq m} S_{ij}^\alpha, \min_{1\leq j\leq m}\max_{1\leq i\leq m_q} S_{ij}^\alpha\right)\right.\right. \\
&\quad \left.\left. - \min\left(\min_{1\leq i\leq m_q}\max_{1\leq j\leq m} S_{ij}^\beta, \min_{1\leq j\leq m}\max_{1\leq i\leq m_q} S_{ij}^\beta\right) > 0 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&\geq \Pr\left(\forall_{\alpha,\beta}\left(\min S_{ij}^\alpha - \Delta_2 - (\max S_{ij}^\beta + \Delta_1) > 0 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&= \Pr\left(\forall_{\alpha,\beta}\left(\min S_{ij}^\alpha - \max S_{ij}^\beta > \Delta_1+\Delta_2 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&= \Pr\left(\forall_{\alpha,\beta}\left(B_\beta - B_\alpha^\star > \Delta_1+\Delta_2 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&\geq \Pr\left(\forall_{\alpha,\beta}\left(B' - B^\star > \Delta_1+\Delta_2 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&\geq \Pr\left(\forall_{\alpha,\beta}\left(2(\Delta_1+\Delta_2) > \Delta_1+\Delta_2 \mid \mathbb{1}=1\right)\right)\Pr(\mathbb{1}=1) \\
&= 1 * \Pr(\mathbb{1}=1) \geq 1 - ((n-k) * \frac{\delta}{2(n-k)} + \frac{\delta}{2k} * k) \\
&= 1 - \delta.
\end{aligned}$$

Therefore, with this choice of L, the algorithm effectively solves the approximate top-k vector set search problem. By eliminating the data-dependent correlation terms $\gamma_{\max}$ and $\xi_{\max}$, and considering the practical scenario where $n \gg k$, then $L = O\left(\log\left(\frac{nm_qm}{\delta}\right)\right)$. □

C. Performance Discussion

The time complexity of `BioVSS` is $O(nm^2L/w)$, where n is the number of vector sets, m is the number of vectors per set, L is the binary vector length, and w is the machine word size (typically 32 or 64 bits). The factor L/w represents the number of machine words needed to store each binary vector, allowing for parallel processing of w bits. This results in improved efficiency compared to real-number operations.

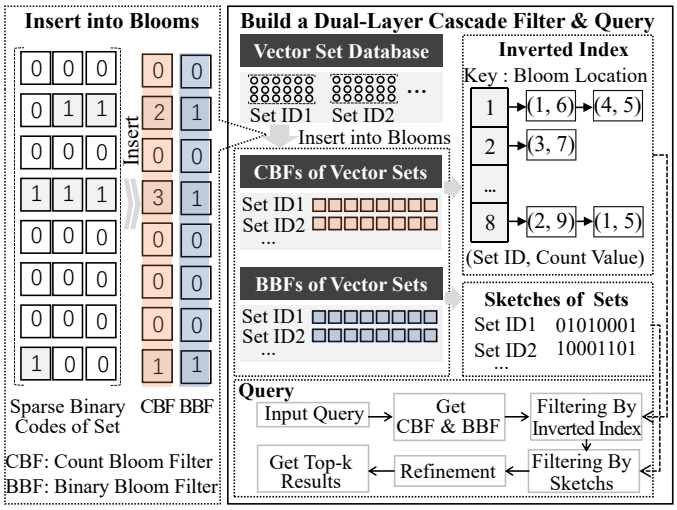


Figure 5: Filter Construction and Query Execution in `BioVSS++`

Despite this optimization, `BioVSS` still requires an exhaustive scan. To address this limitation, we propose enhancing `BioVSS` with a filter. Next, we will detail this method.

V. Enhancing BioVSS via Cascade Filter

We propose `BioVSS++`, an enhanced method that addresses the linear scan limitation of `BioVSS` through a Bio-Inspired Dual-Layer Cascade Filter (`BioFilter`). The fundamental principle of `BioVSS++` exploits the structural similarity between `BioHash` and Bloom filters, both employing sparse structure for vector encoding and set storage. Next, we detail the filter construction and the search execution.

A. Filter Construction

The bio-inspired dual-layer cascade filter forms the basis for efficient search execution in `BioVSS++`. `BioFilter` consists of two key components: an inverted index based on count Bloom filters and vector set sketches based on binary Bloom filters. The inverted index leverages the count Bloom filter to construct inverted lists, each storing vector sets based on their count values. The vector set sketches employ a binary Bloom filter for second filtering via Hamming distance, enabling efficient scanning. `BioFilter`'s dual-layer approach significantly enhances search efficiency by reducing the candidate set through successive refinement stages.

1) ***Inverted Index Based on Count Bloom Filter****:* the inverted index serves as the first layer of `BioFilter`. It accelerates search execution by constructing an inverted index of vector sets to reduce candidate items.

To mitigate the linear scanning limitation of Algorithm 1, we leverage the count Bloom filters to manage vector sets, which form the basis for constructing the inverted index.

Definition 8 (**Count Bloom Filter** [5])**.** *Let* $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m\}$ *be a vector set. A* ***count Bloom filter*** $\mathbf{C} = (c_1, c_2, ..., c_b)$ *for* $\mathbf{V}$ *is an array of* b *counters, where each counter* c_i *represents the sum of the* i*-th bits of the sparse*

Algorithm 3: `Gen_Count_Bloom_Filter`($\mathbf{D}, L_{wta}, b$)

Input: $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^{n}$: vector set database, L_{wta}: # of `WTA`, b: the length of bloom filters
Output: C: count Bloom filters

```
// Generate binary codes: Algorithm 1
```
1 $\mathbf{D^H} = $ `Gen_Binary_Codes`($\mathbf{D}, L_{wta}$);
```
// Construct Count Bloom Filters
```
2 $C = \{\mathbf{C}^{(j)}\}_{j=1}^{n}$, where $\mathbf{C}^{(j)} \in \mathbb{N}^b$;
3 **for** $j = 1$ **to** n **do**
4 $\quad \mathbf{C}^{(j)} = \sum_{\mathbf{h} \in \mathbf{H}_j} \mathbf{h}$, where $\mathbf{H}_j \in \mathbf{D^H}$;
5 **return** C;

binary codes of vectors in $\mathbf{V}$*. Formally,* $c_i = \sum_{j=1}^{m} \mathcal{H}(\mathbf{v}_j)_i$*, where* $\mathcal{H}(\mathbf{v}_j)_i$ *is the i-th bit of the binary code of* $\mathbf{v}_j$*.*

The count Bloom filter encodes each binary code's frequency across all vectors in the set. Next, we define the inverted index, which is built upon the count Bloom filters.

Definition 9 (**Inverted Index Based on Count Bloom Filter**)**.** *Let* $\mathbf{D} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$ *be a database of vector sets. A* ***inverted index based on count bloom filter*** $\mathbf{I}$ *is a data structure that maps each bit position to a sorted list of tuples. For each position* $i \in \{1, 2, ..., b\}$*,* $\mathbf{I}[i]$ *contains a list of tuples* $(j, c_i^{(j)})$*, where* j *is the index of the vector set* $\mathbf{V}_j \in \mathbf{D}$*, and* $c_i^{(j)}$ *is the i-th count value in the count Bloom filter for* $\mathbf{V}_j$*. The list is sorted in descending order based on* $c_i^{(j)}$ *values.*

Algorithm 4: `Build_Inverted_Index`($\mathbf{D}, L_{wta}, b$)

Input: $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^{n}$: vector set database, L_{wta}: # of `WTA`, b: the length of bloom filters
Output: $\mathbf{I}$: inverted index

```
// Generate Count Bloom Filters: Algorithm 3
```
1 $C = $ `Gen_Count_Bloom_Filter`($\mathbf{D}, L_{wta}, b$);
```
// Build Inverted Index
```
2 Initialize $\mathbf{I} = \{\mathbf{I}_i\}_{i=1}^{b}$, where $\mathbf{I}_i = \emptyset$ for $i = 1, \ldots, b$;
3 **for** $j = 1$ **to** n **do**
4 $\quad$ **for** $i = 1$ **to** b **do**
5 $\quad\quad c_i^{(j)} = \mathbf{C}^{(j)}[i]$;
6 $\quad\quad \mathbf{I}_i = \mathbf{I}_i \cup \{(j, c_i^{(j)})\}$;
7 **for** $i = 1$ **to** b **do**
8 $\quad$ Sort $\mathbf{I}_i$ in descending order by $c_i^{(j)}$;
9 **return** $\mathbf{I}$;

The inverted index facilitates efficient vector set search. As shown in Figure 5, each vector set is inserted into count Bloom filters. Subsequently, inverted lists are constructed and sorted in descending order based on the count values at various positions in these filters. This approach capitalizes on the principle that higher count values indicate a greater likelihood of collisions at the same positions for similar vector sets.

Algorithm 3 describes the process of generating count Bloom filters for all vector sets. The input consists of a database $\mathbf{D}$ of n vector sets, the winner-takes-all parameter L_{wta}, and the length b of the bloom filters. The algorithm begins by generating sparse binary codes for the database using the `Gen_Binary_Codes` function from Algorithm 1 (line 1). It then constructs count Bloom filters C for each vector set. Each count Bloom filter $\mathbf{C}^{(j)}$ is created by summing the binary codes of all vectors in the corresponding set $\mathbf{H}_j$ (line 2-4) and then returns count Bloom filters (line 5).

Algorithm 5: `Build_Set_Sketches`($\mathbf{D}, L_{wta}, b$)

Input: $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^{n}$: vector set database, L_{wta}: # of `WTA`, b: the length of bloom filters
Output: $\mathcal{S} = \{\mathbf{S}^{(i)}\}_{i=1}^{n}$: set sketches

```
// Generate Binary Codes: Algorithm 1
```
1 $\mathbf{D^H} = $ `Gen_Binary_Codes`($\mathbf{D}, L_{wta}$);
```
// Construct Binary Bloom Filters
```
2 $\mathcal{B} = \{\mathbf{B}^{(i)}\}_{i=1}^{n}$, where $\mathbf{B}^{(i)} \in \{0, 1\}^b$;
3 **for** $i = 1$ **to** n **do**
4 $\quad \mathbf{B}^{(i)} = \bigvee_{\mathbf{h} \in \mathbf{H}_i} \mathbf{h}$, where $\mathbf{H}_i \in \mathbf{D^H}$; `// ∨ is OR`
```
// Build Set Sketches
```
5 $\mathcal{S} = \{\mathbf{S}^{(i)}\}_{i=1}^{n}$, where $\mathbf{S}^{(i)} = \mathbf{B}^{(i)}$;
6 **return** $\mathcal{S}$;

Algorithm 4 describes the process of building an inverted index. The input is the same as algorithm 3. The output is an inverted index $\mathbf{I}$. First, count Bloom filters for all data are generated through Algorithm 3 (line 1). For each position i in the count Bloom filters, the algorithm creates an inverted list $\mathbf{I}_i$ containing pairs of set indices and their corresponding count values (lines 2-6). Finally, each inverted list is sorted in descending order based on the count values (lines 7-8). This inverted index structure effectively reduces the search space.

2) ***Vector Set Sketches Based on Binary Bloom Filter****:* vector set sketches form the second layer of `BioFilter`, providing a single binary representation of each vector set. These sketches are based on binary Bloom filters, constructed by applying a bitwise OR operation to the binary codes of all vectors within each set.

The binary nature of these sketches enables efficient similarity estimation through Hamming distance, leveraging fast XOR and popcount operations available in modern CPUs [29]. The sketches reduce computational complexity by avoiding the expensive aggregation operations in Hausdorff distance.

To better understand the operation of the building set sketches, we first present the binary Bloom filter:

Definition 10 (**Binary Bloom Filter** [5])**.** *Let* $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m\}$ *be a set of vectors. The* ***binary Bloom filter*** $\mathbf{B}$ *for* $\mathbf{V}$ *is a binary array of length b, obtained by performing a bitwise OR operation on the binary codes all vectors in* $\mathbf{V}$*. Formally,* $\mathbf{B} = \mathcal{H}(\mathbf{v}_1) \vee \mathcal{H}(\mathbf{v}_2) \vee ... \vee \mathcal{H}(\mathbf{v}_m)$*, where* $\mathcal{H}(\mathbf{v}_i)$ *is the binary vector of* $\mathbf{v}_i$ *and* $\vee$ *denotes the bitwise OR operation.*

As shown in Figure 5, each vector set is inserted into binary Bloom filters. Subsequently, the binary Bloom filter of each vector set is transformed into a vector set sketch.

Algorithm 5 outlines the process of building set sketches in vector set databases. The input consists of a database $\mathbf{D}$ of n vector sets, the winner-takes-all parameter L_{wta}, and the length b of the binary code. The output is a collection of set sketches $\mathcal{S}$ that provide compact representations of the vector sets. The algorithm begins by generating sparse binary codes for the database using the `Gen_Binary_Codes` function

Algorithm 6: `BioVSS++_Topk_Search`$(\mathbf{Q}, k, \mathbf{D}, \mathbf{I}, \mathcal{S}, \Theta)$

Input: $\mathbf{Q}$: query vector set, k: # of results, $\mathbf{D} = \{\mathbf{V}_i\}_{i=1}^{n}$: vector set database, $\mathbf{I}$: inverted index, $\mathcal{S} = \{\mathbf{S}^{(i)}\}_{i=1}^{n}$: set sketches, $\Theta = \{A$: access number of lists, M : minimum count, c : the size of candidate set, T : # of candidates, L_{wta} : # of `WTA`, b : the length of bloom filters$\}$

Output: $\mathcal{R}$: top-k vector sets

```
// Get Query Count Bloom Filter and
   Sketch: Algorithm 3 and 5
```
1 $\mathbf{C}_Q$ = `Gen_Count_Bloom_Filter`$(\mathbf{Q})$;
2 $\mathbf{S}_Q$ = `Build_Set_Sketch`$(\mathbf{Q}, L_{wta}, b)$;
```
// Filtering by Inverted Index
```
3 π = `ArgsortDescending`$(\mathbf{C}_Q)$;
4 $\mathcal{P} = \{\pi(i) : i \in [1, A]\}$;
5 $\mathcal{F}_1 = \emptyset$;
6 **for** $p \in \mathcal{P}$ **do**
7 **for** $(i, c_p^{(i)}) \in \mathbf{I}[p]$ **do**
8 **if** $c_p^{(i)} \geq M$ **then**
9 $\mathcal{F}_1 = \mathcal{F}_1 \cup \{i\}$;
```
// Filtering by Sketches
```
10 Initialize $\mathcal{G} \leftarrow$ empty max-heap with capacity c;
11 **for** $i \in \mathcal{F}_1$ **do**
12 d = `Hamming`$(\mathbf{S}_Q, \mathbf{S}^{(i)})$;
13 **if** $|\mathcal{G}| < T$ **then**
14 $\mathcal{G}$.`push`$((d, i))$;
15 **else if** $d < \mathcal{G}.top()[0]$ **then**
16 $\mathcal{G}$.`pop`();
17 $\mathcal{G}$.`push`$((d, i))$;
18 $\mathcal{F}_2 = \{i | (\_, i) \in \mathcal{G}\}$;
```
// Select top-k results
```
19 $\mathcal{D} = \emptyset$;
20 **for** *each* $(i, d_H) \in \mathcal{F}$ **do**
21 $d_i = Haus(\mathbf{Q}, \mathbf{V}_i)$;
22 $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{V}_i, d_i)\}$;
23 $\mathbf{R} = \{(\mathbf{V}_i, d_i) \in \mathcal{D} \mid d_i \leq d_i^{(k)}\}$, where $d_i^{(k)}$ is the k-th smallest d_i in $\mathcal{D}$;
24 **return** $\mathbf{R}$;

from Algorithm 1 (line 1). It then constructs binary Bloom filters $\mathcal{B}$ for each vector set (lines 2-4). Each binary Bloom filter $\mathbf{B}^{(i)}$ is created by applying a bitwise OR operation ($\bigvee$) to all binary codes of vectors in the corresponding set $\mathbf{H}_i$. Finally, the algorithm builds the set sketches $\mathcal{S}$ (line 5) by directly using the binary Bloom filters as the sketches and then returns $\mathcal{S}$.

B. Search Execution in `BioVSS++`

This section introduces the search execution process in `BioVSS++`. Building upon the inverted index and set sketches, `BioFilter` can quickly filter unrelated vector sets.

The strategy of our search execution lies in a dual-layer filtering mechanism. In the first layer, we employ the inverted index to quickly eliminate a large portion of dissimilar vector sets. In the second layer, we use the vector set sketches for further refinement. This overcomes the limitations of linear scanning in Algorithm 1, reducing the computational overhead.

Algorithm 6 presents `BioVSS++` top-k query process, which employs a two-stage filter `BioFilter` for efficient vector set search. The input includes a query set $\mathbf{Q}$, the number of desired results k, the database $\mathbf{D}$, a pre-computed inverted index $\mathbf{I}$, set sketches $\mathcal{S}$, and parameters Θ. The output $\mathbf{R}$ contains the top-k most similar vector sets. The algorithm begins by generating a count Bloom filter $\mathbf{C}_Q$ and a set sketch $\mathbf{S}_Q$ for the query set (lines 1-2). `BioFilter` then applies two filtering stages: 1) inverted index filtering (lines 3-9): This stage uses the query's count Bloom filter to identify potential candidates. It selects the top-A positions with the highest counts in $\mathbf{C}_Q$ and retrieves vector sets from the inverted index $\mathbf{I}$ that have counts above a threshold M at these positions. This produces an initial candidate set $\mathcal{F}_1$. 2) sketch-based filtering (lines 10-18): This stage refines the candidates using set sketches. It computes the Hamming distance between the query sketch $\mathbf{S}_Q$ and each candidate's sketch $\mathbf{S}^{(i)}$, maintaining a max-heap of the T closest candidates. This results in a further refined candidate set $\mathcal{F}_2$. Finally, the algorithm computes the actual Hausdorff distance between $\mathbf{Q}$ and each remaining candidate in $\mathcal{F}_2$ (lines 19-22), selecting the top-k results based on these distances (line 23) and then return it (line 24).

C. Discussion on Metrics Extensibility

`BioVSS++` algorithm exhibits potential compatibility with diverse set-based distance metrics beyond the Hausdorff distance, owing to the decoupling of the filter structure from the specific distance metric employed. Examples of potentially compatible distance metrics include: 1) set distances based on maximum or minimum point-pair distances [41]; 2) mean aggregate distance [13]; 3) Hausdorff distance variants [7], etc.

Specifically, the low coupling between distance metrics and the filter is evident in the construction process. Whether building inverted indexes or hash codes, the specific distance metric is not involved. This allows `BioVSS++` to be potentially extended to other metrics. Further exploration can be found in Appendix VII-B4 and VII-C.

VI. Experiments

A. Settings

1) Datasets: the text datasets for this work are sourced from the Microsoft academic graph [39]. We extracted two datasets from the fields of computer science and medicine, including those with at least two first-author papers. The texts from these papers were converted into vectors using the embedding model all-MiniLM-L6-v2[1] (commonly used). Additionally, we constructed an image dataset using ResNet18[1] for feature extraction. Datasets' details are shown as follows:

- Computer Science Literature (CS): It contains 1,192,792 vector sets in the field of computer science, with 5,553,031 vectors. Each set comprises 2 to 362 vectors (Table II).

[1]https://www.sbert.net & https://huggingface.co

TABLE II: Summary of Datasets

Dataset	# of Vectors	# of Vector Sets	Dim.	# of Vectors/Set
CS	5,553,031	1,192,792	384	[2, 362]
Medicine	15,053,338	2,693,842	384	[2, 1923]
Picture	2,513,970	982,730	512	[2, 9]

- Medicine Literature (Medicine): It contains 2,693,842 vector sets in the field of Medicine, with 15,053,338 vectors. Each set comprises 2 to 1923 vectors (Table II).
- Product Pictures (Picture): It contains 982,730 vector sets sourced from the AliProduct dataset [40], with 2,513,970 vectors. Each set represents 2 to 9 images of the same product, covering 50,000 different products (Table II).

2) Baselines: we compare our proposed method against several indexing and quantization techniques. Specifically, we employ methods from the Faiss library [10] developed by Facebook as comparative baselines. The methods evaluated include: 1) `IVFFLAT` [49], [10], using an inverted file index with flat vectors for efficient high-dimensional data processing; 2) `IndexIVFPQ` [14], [20], [10], combining inverted file index with product quantization for vector compression and improved query speed; 3) `IVFScalarQuantizer` [10], employing scalar quantization within the inverted file index to optimize speed-accuracy trade-off; 4) `BioVSS++`, our proposed method utilizing a bio-inspired dual-layer cascade filter to enhance efficiency through effective pruning.

Due to the absence of efficient direct vector set search methods using Hausdorff distance in high-dimensional spaces, all methods rely on centroid vectors to construct indices.

3) Evaluation Metric: to evaluate the performance of vector set search algorithms, we employ the recall rate at different top-k values as the evaluation metric.

The recall rate at top-k is defined as follows: $\text{Recall@}k = \frac{|R_k(\mathbf{Q}) \cap G_k(\mathbf{Q})|}{|G_k(\mathbf{Q})|}$, where $\mathbf{Q}$ denotes a query set, $R_k(\mathbf{Q})$ represents algorithm's the top-k retrieved results for query $\mathbf{Q}$, and $G_k(\mathbf{Q})$ denotes the ground-truth (accurate calculations by Definition 4). We perform 500 queries and report the average recall rate.

4) Implementation: Our experiments were conducted on a computing platform with Intel Xeon Platinum 8352V and 512 GB of memory. The core components of our method[2] were implemented in C++ and interfaced with Python.

5) Default Parameters: In our experiments, we focus on the following parameters: 1) the size of Bloom filter $\{\underline{1024}, 2048\}$; 2) the number of winner-takes-all $\{16, 32, 48, \underline{64}\}$; 3) the list number of inverted index accessed $\{1, 2, \underline{3}\}$; 4) Minimum count value of inverted index $\{\underline{1}, 2\}$; 5) The size of candidate set $\{20k, 30k, 40k, \underline{50k}\}$; and 6) The number of results returned $\{\underline{3}, \underline{5}, 10, 15, 20, 25, 30\}$. Underlined values denote default parameters in our controlled experiments.

B. Storage and Construction Efficiency of Filter Structures

`BioFilter` comprises two sparse data structures: the count Bloom filter and the binary Bloom filter. Both filters exhibit sparsity, necessitating optimized storage strategies.

The storage optimization leverages two established sparse formats: Coordinate (`COO`) [4] and Compressed Sparse Row (`CSR`) [4]. `COO` format manages dynamic updates through (row, column, value) tuples. `CSR` format achieves superior compression by maintaining row pointers and column indices, which is particularly effective for static index structures.

[2]https://github.com/whu-totemdb/biovss

TABLE III: Filter Storage Comparison on CS Dataset

Bloom	L	Count Bloom Space (GB)			Binary Bloom Space (GB)		
		Dense	COO	CSR	Dense	COO	CSR
1024	16	9.1	1.09	0.55	1.14	0.36	0.19
	32		2.02	1.01		0.67	0.34
	48		2.84	1.43		0.95	0.48
	64		3.59	1.8		1.2	0.6
2048	16	18.2	1.18	0.59	2.28	0.39	0.2
	32		2.2	1.1		0.73	0.37
	48		3.13	1.57		1.04	0.53
	64		4	2		1.33	0.67

Table III demonstrates the results on CS. With a 1024-size Bloom filter, `CSR` reduces the storage overhead from 9.1GB to 0.55GB for the count Bloom filter, achieving a 94% reduction ratio. Similar patterns emerge across different parameter settings, with `CSR` consistently outperforming `COO` in storage efficiency. Results for Medicine and Picture exhibit analogous characteristics and are detailed in the Appendix VII-B3. Table IV illustrates the time overhead of different stages. While `BioHash` training constitutes the primary computational cost at $1504s$, the construction of count and binary Bloom filters demonstrates efficiency, requiring only $24s$ and $22s$.

TABLE IV: Filter Processing Time on CS Dataset

Processing Stage	`BioHash` Training	`BioHash` Hashing	Count Bloom	Single Bloom
Time	1504s	14s	24s	22s

C. Performance Analysis and Parameter Experiments

In this section, we first conduct preliminary experiments to evaluate the performance of `BioVSS` and `BioVSS++`, demonstrating their superiority over brute-force search. We then concentrate analysis on `BioVSS++`, an enhanced version that incorporates `BioFilter` for improved efficiency. Finally, we perform extensive parameter tuning studies on `BioVSS++` to assess its performance and optimize its configuration.

1) Comparison with Brute-Force Search: We first compare the performance of `BioVSS` and `BioVSS++` against brute-force search in terms of execution time and recall rate on CS, Medicine, and Picture datasets, with a candidate set size of $20k$. As shown in Tables V, VI and VII, both `BioVSS` and `BioVSS++` significantly outperform brute-force search in

TABLE V: Speedup vs. Linear Scan on CS Dataset

Method	Total Time (s)	Speedup	Top-3 Recall	Top-5 Recall
`Brute`	9.16	1x	100%	100%
`BioVSS`	0.73	12x	97.5%	97.2%
`BioVSS++`	0.20	46x	97.9%	96.2%

TABLE VI: Speedup vs. Linear Scan on Medicine Dataset

Method	Total Time (s)	Speedup	Top-3 Recall	Top-5 Recall
`Brute`	16.32	1x	100%	100%
`BioVSS`	2.13	8x	96.5%	95.8%
`BioVSS++`	0.24	78x	93.8%	92.3%

TABLE VII: Speedup vs. Linear Scan on Picture Dataset

Method	Total Time (s)	Speedup	Top-3 Recall	Top-5 Recall
`Brute`	8.75	1x	100%	100%
`BioVSS`	0.49	17x	100%	99.9%
`BioVSS++`	0.20	44x	97.8%	96.1%

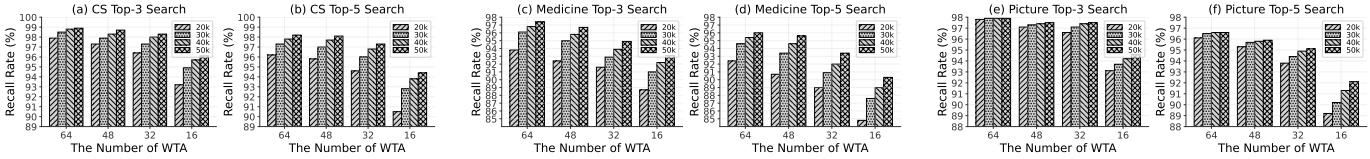


Figure 6: Recall Rate with Winner-Take-All Number (Bloom Size = 1024)

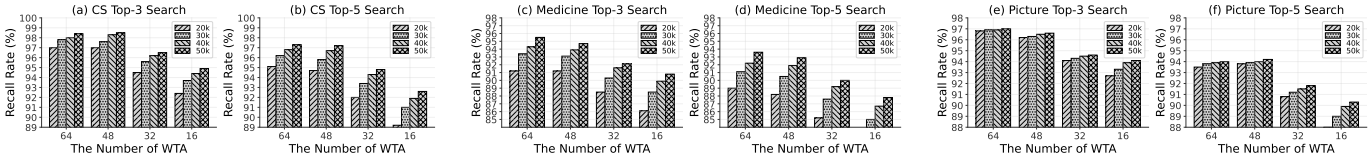


Figure 7: Recall Rate with Winner-Take-All Number (Bloom Size = 2048)

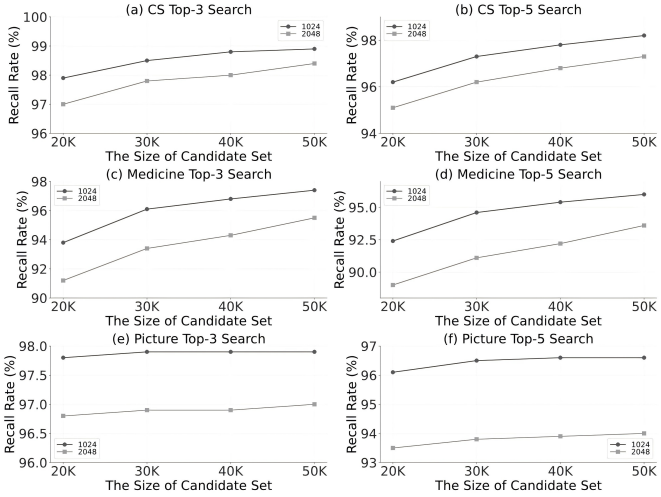


Figure 8: Impact of Bloom Filter Size on Recall

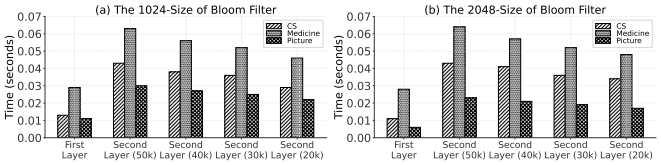


Figure 9: Filtering Times by the Size of Bloom Filter

terms of speed, while maintaining high recall rates. On CS dataset, BioVSS++ achieves a remarkable 46-fold speedup compared to brute-force search, reducing the total execution time from 9.16 seconds to just 0.20 seconds, while still maintaining high top-3 and top-5 recall rates of 97.9% and 96.2%. The performance gain is even more pronounced on Medicine dataset, where BioVSS++ demonstrates a 78 times speedup, reducing the execution time from 16.32 seconds to a mere 0.24 seconds. On Picture dataset, BioVSS++ achieves a 44-fold speedup, reducing execution time from 8.75 seconds to 0.20 seconds, with recall rates of 97.8% for top-3 and 96.1% for top-5. This substantial improvement can be attributed to the filtering mechanism employed by BioVSS++, which effectively addresses the limitation of global scanning present in BioVSS.

2) Parameter Study of BioVSS++: To optimize the performance of BioVSS++, we conducted a comprehensive parameter study. We examined several key parameters: the number of winner-takes-all, the size of the Bloom filter, the list number of inverted index accessed, and the minimum count value of the inverted index.

Number of Winners-Take-All. The number of winner-takes-all (see Definition 7) in BioVSS++ corresponds to the number of hash functions. As shown in Figures 6 and 7, increasing this parameter from 16 to 48 led to significant performance improvements across CS, Medicine, and Picture datasets for Bloom filter sizes of 1024 and 2048. On CS dataset, with a 1024-size Bloom filter and 20k candidates, recall improved by 5.7% when increasing the parameter from 16 to 48. However, the performance gain plateaued between 48 and 64, with only a 0.4% improvement in the same scenario. Similarly, Medicine and Picture datasets exhibit the same trend when increasing the parameter. This trend was consistent across different datasets and Bloom filter sizes. This improvement can be attributed to the enhanced distance-preserving property of the hash code as the number of winner-takes-all increases. Hence, the default number of winner-takes-all is set to 64.

Size of the Bloom Filter. The size of the Bloom filter directly influences the length of set sketches and the list number in the inverted index for BioVSS++. As illustrated in Figure 8, our experiments explore filter sizes of 1024 and 2048, revealing that a Bloom filter with a size of 1024 achieves optimal recall rates across all candidate numbers. On CS dataset, the 1024-size filter configuration yields a 98.9% recall rate with a candidate set of 50k, while maintaining a robust 98% recall even with a reduced set of 20k candidates. Similarly, for Medicine and Picture datasets, the 1024-size filter achieves optimal recall rates. The effectiveness of this filter size can be attributed to its capacity to capture discriminative features without over-emphasizing local characteristics, thereby providing a good trade-off between specificity and generalization in this process. As demonstrated in Figure 9, The various sizes of the Bloom filters exhibit low latency below 70 milliseconds. Hence, the default size of the Bloom filter is set to 1024.

List Number of Inverted Index Accessed. The list number of inverted index accessed determines the search range in BioVSS, with larger values leading to broader searches. As shown in Table VIII, increasing this parameter from 1 to 3 significantly improves recall rates while modestly increasing processing time. For CS dataset with a 1024-size Bloom filter, the top-3 recall improves from 92.9% to 98.9%, and the top-5 recall from 91.1% to 98.2%, with processing time increasing from 0.008s to 0.013s. Similar improvements are observed

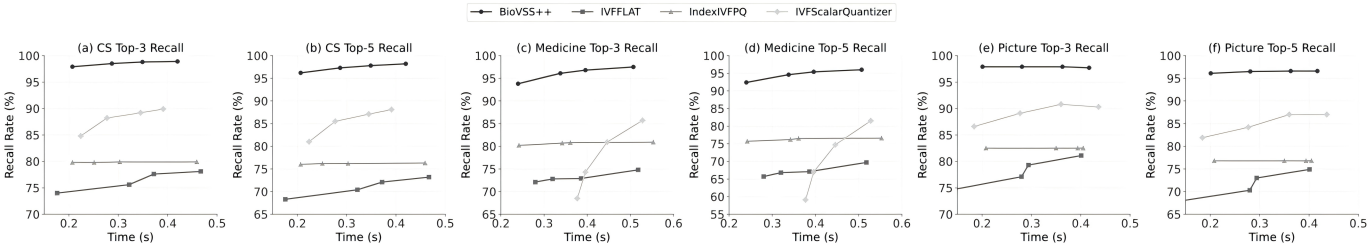


Figure 10: Recall Rate Comparison for Different Methods

TABLE VIII: List Access Number for Top-3 (T-3) and Top-5 (T-5)

Dataset	1 Access			2 Access			3 Access		
	Recall (%)		Time	Recall (%)		Time	Recall (%)		Time
	T-3	T-5	(s)	T-3	T-5	(s)	T-3	T-5	(s)
CS (1024)	92.9	91.1	.008	98.0	97.1	.012	98.9	98.2	.013
CS (2048)	91.9	89.4	.008	97.6	96.4	.011	98.4	97.3	.011
Medicine (1024)	92.8	90.8	.014	96.9	95.6	.018	97.4	96.0	.029
Medicine (2048)	90.3	86.8	.014	94.3	92.2	.019	95.5	93.6	.028
Picture (1024)	85.1	76.3	.017	94.8	91.2	.024	97.9	96.6	.030
Picture (2048)	81.5	73.6	.013	93.1	88.0	.018	97.0	94.0	.023

TABLE IX: Minimum Count Value with Recall Rate and Time

Dataset	Minimum Count = 1			Minimum Count = 2		
	Top-3	Top-5	Total Time	Top-3	Top-5	Total Time
CS (1024)	98.9%	98.2%	0.42s	96.5%	95.1%	0.45s
CS (2048)	98.4%	97.3%	0.45s	95.1%	94.0%	0.44s
Medicine (1024)	97.4%	96.0%	0.51s	93.3%	91.8%	0.50s
Medicine (2048)	95.5%	93.6%	0.48s	89.7%	87.0%	0.47s
Picture (1024)	97.9%	96.6%	0.42s	92.0%	87.3%	0.45s
Picture (2048)	97.0%	94.0%	0.43s	87.9%	81.6%	0.43s

with a 2048-size Bloom filter. Medicine and Picture datasets show the same trends, with slightly lower recall rates but similar time increases. Notably, the recall rate gain diminishes when moving from 2 to 3 accesses. Hence, the default list number of inverted index accessed is set to 3.

Minimum Count Value of Inverted Index. The minimum count value of the inverted index determines the threshold for accessing items in the inverted index lists. As illustrated in Figure IX, this parameter significantly impacts the trade-off between recall. Setting the value to 0 requires traversing all items. A value of 1 leverages the hash codes' sparsity, eliminating most irrelevant items while maintaining high recall. Increasing to 2 further reduces candidates but at the cost of the recall rate. For CS dataset with a 1024-size Bloom filter, the top-3 recall decreases from 98.9% at count 1 to 96.5% at count 2. Top-5 recall shows a similar trend, declining from 98.2% to 95.7%. Medicine and Picture datasets exhibit similar trends, with comparable drops in top-3 and top-5 recall rates. Thus, we set the default minimum count to 1.

D. Comparison Experiment

To evaluate `BioVSS++` method, we conducted comparative experiments against baseline algorithms on CS, Medicine, and Picture datasets. We analyzed query time and recall rate. The following presents the results and discussion.

Figures 10(a) and (b) present the experimental results on CS dataset. By comparing the trends of the time-recall curves, it is evident that our proposed `BioVSS++` algorithm outperforms the other three baseline algorithms. To control the query time, we adjust the size of the candidate set. The experimental results demonstrate that the recall rates of all algorithms increase with the growth of query time. Specifically, On `BioVSS++` achieves a recall rate of 98.9% in just 0.2 seconds for the top-3 scenario, and the recall rate further reaches 98.2% at 0.47 seconds. Even in the top-5 case, the recall rate of `BioVSS++` remains above 90%. It is worth noting that the recall rates of `IVFScalarQuantizer` and `IVFFLAT` exhibit a significant upward trend as the query time increases, while the upward trend of `IndexIVFPQ` is relatively less pronounced. We speculate that this may be due to the limitations of the product quantization encoding scheme. Product quantization encoding compresses data by dividing high-dimensional vectors into multiple subvectors and quantizing each subvector. However, this encoding approach may lead to information loss, thereby affecting the room for improvement in recall rate.

Figures 10(c) and (d) showcase the query efficiency on Medicine dataset. Due to the larger scale, the query time increases accordingly. The recall rate of the `IVFScalarQuantizer` algorithm demonstrates a more significant upward trend as the query time increases. This indicates that for large-scale datasets, the `IVFScalarQuantizer` algorithm may be more sensitive to the candidate set size. Concurrently, this also implicitly reflects that our `BioVSS++` algorithm maintains good query performance even with smaller candidate set sizes. Figures 10(e) and (f) showcase the query efficiency on Picture dataset, exhibiting similar trends to CS dataset.

VII. Conclusions

In this paper, we investigated the relatively unexplored problem of vector set search. We introduced `BioVSS` and `BioVSS++` for efficient vector set search using the Hausdorff distance. We provide a theoretical analysis of algorithm correctness. Our dual-layer filter effectively prunes irrelevant items, reducing computational overhead. Experiments show that the proposed method achieves over a 50 times speedup compared to traditional linear scanning methods on million-scale datasets, with a recall rate of up to 98.9%. Future work will extend our framework to support more distance metrics.

Acknowledgement

This work was supported by the National Key R&D Program of China (2023YFB4503600), the National Natural Science Foundation of China (62202338, 62372337), and the Key R&D Program of Hubei Province (2023BAB081)

REFERENCES

[1] M. D. Adelfio, S. Nutanong, and H. Samet. Similarity search on a large collection of point sets. In *SIGSPATIAL*, pages 132–141, 2011.

[2] M. J. Atallah. A linear time algorithm for the hausdorff distance between convex polygons. *Information Processing Letters*, 17(4):207–209, 1983.

[3] M. Aumüller and M. Ceccarello. Solving k-closest pairs in high-dimensional data. In *SISAP*, volume 14289, pages 200–214, 2023.

[4] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC*. ACM, 2009.

[5] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *ESA*, volume 4168, pages 684–695, 2006.

[6] L. Buck and R. Axel. A novel multigene family may encode odorant receptors: a molecular basis for odor recognition. *Cell*, 65(1):175–187, 1991.

[7] A. Conci and C. S. Kubrusly. Distance between sets-a survey. *Advances in Mathematical Sciences and Applications*, 26(1):1–18, 2018.

[8] S. Dasgupta, C. F. Stevens, and S. Navlakha. A neural algorithm for a fundamental computing problem. *Science*, 358(6364):793–796, 2017.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[10] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *CoRR*, abs/2401.08281, 2024.

[11] J. Engels, B. Coleman, V. Lakshman, and A. Shrivastava. DESSERT: an efficient algorithm for vector set search with vector set queries. In *NeurIPS*, 2023.

[12] Y. Fu, C. Chen, X. Chen, W. Wong, and B. He. Optimizing the number of clusters for billion-scale quantization-based nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering*, (01):1–14, 2024.

[13] O. Fujita. Metrics based on average distance between sets. *Japan Journal of Industrial and Applied Mathematics*, 30:1–19, 2013.

[14] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2014.

[15] G. Gupta, T. Medini, A. Shrivastava, and A. J. Smola. BLISS: A billion scale index using iterative re-partitioning. In *SIGKDD*, pages 486–495, 2022.

[16] F. Hausdorff. *Grundzüge der mengenlehre*, volume 7. von Veit, 1914.

[17] J. Henrikson. Completeness and total boundedness of the hausdorff metric. *MIT Undergraduate Journal of Mathematics*, 1(69-80):10, 1999.

[18] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338, 2013.

[19] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[20] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[21] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[22] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[23] M. Leybovich and O. Shmueli. Efficient approximate search for sets of lineage vectors. In *TaPP*, pages 5:1–5:8. ACM, 2022.

[24] W. Li, C. Feng, D. Lian, Y. Xie, H. Liu, Y. Ge, and E. Chen. Learning balanced tree indexes for large-scale vector retrieval. In *KDD*, pages 1353–1362, 2023.

[25] 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*, 32(8):1475–1488, 2020.

[26] V. W. Liang, Y. Zhang, Y. Kwon, S. Yeung, and J. Y. Zou. Mind the gap: Understanding the modality gap in multi-modal contrastive representation learning. *NeurIPS*, 35:17612–17625, 2022.

[27] S. X. Luo, R. Axel, and L. Abbott. Generating sparse and selective third-order responses in the olfactory system of the fly. *Proceedings of the National Academy of Sciences*, 107(23):10713–10718, 2010.

[28] 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*, 42(4):824–836, 2020.

[29] W. Muła, N. Kurz, and D. Lemire. Faster population counts using avx2 instructions. *The Computer Journal*, 61(1):111–120, 2018.

[30] S. Mysore, M. Jasim, A. Mccallum, and H. Zamani. Editable user profiles for controllable text recommendations. In *SIGIR*, page 993–1003, 2023.

[31] P. Nigam, Y. Song, V. Mohan, V. Lakshman, W. Ding, A. Shingavi, C. H. Teo, H. Gu, and B. Yin. Semantic product search. In *SIGKDD*, pages 2876–2885, 2019.

[32] S. Nutanong, E. H. Jacox, and H. Samet. An incremental hausdorff distance calculation algorithm. *Proceedings of the VLDB Endowment*, 4(8):506–517, 2011.

[33] U. Pace, E. Hanski, Y. Salomon, and D. Lancet. Odorant-sensitive adenylate cyclase may mediate olfactory reception. *Nature*, 316(6025):255–258, 1985.

[34] J. Qin, Y. Wang, C. Xiao, W. Wang, X. Lin, and Y. Ishikawa. Gph: Similarity search in hamming space. In *ICDE*, pages 29–40, 2018.

[35] L. D. Rhein and R. H. Cagan. Biochemical studies of olfaction: isolation, characterization, and odorant binding activity of cilia from rainbow trout olfactory rosettes. *Proceedings of the National Academy of Sciences*, 77(8):4412–4416, 1980.

[36] G. Roh, J. Roh, S. Hwang, and B. Yi. Supporting pattern-matching queries over trajectories on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 23(11):1753–1758, 2011.

[37] C. Ryali, J. Hopfield, L. Grinberg, and D. Krotov. Bio-inspired hashing for unsupervised similarity search. In *ICML*, volume 119, pages 8295–8306, 2020.

[38] B. Schäfermeier, G. Stumme, and T. Hanika. Mapping research trajectories. *CoRR*, abs/2204.11859, 2022.

[39] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. In *WWW*, pages 243–246, 2015.

[40] L. Song, P. Pan, K. Zhao, H. Yang, Y. Chen, Y. Zhang, Y. Xu, and R. Jin. Large-scale training system for 100-million classification at alibaba. In *SIGKDD*, pages 2909–2930. ACM, 2020.

[41] G. T. Toussaint. An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons. *Computing*, 32(4):357–364, 1984.

[42] L. B. Vosshall, A. M. Wong, and R. Axel. An olfactory sensory map in the fly brain. *Cell*, 102(2):147–159, 2000.

[43] M. Wang, H. Wu, X. Ke, Y. Gao, X. Xu, and L. Chen. An interactive multi-modal query answering system with retrieval-augmented large language models. *CoRR*, abs/2407.04217, 2024.

[44] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong. A survey on trajectory data management, analytics, and learning. *ACM Computing Surveys*, 54(2):39:1–39:36, 2022.

[45] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin. Torch: A search engine for trajectory data. In *SIGIR*, pages 535–544. ACM, 2018.

[46] S. Wang, Y. Sun, and Z. Bao. On the efficiency of k-means clustering: Evaluation, optimization, and algorithm selection. *Proceedings of the VLDB Endowment*, 14(2):163–175, 2020.

[47] T. Wei, R. Alkhoury Maroun, Q. Guo, and B. Webb. Devfly: Bio-inspired development of binary connections for locality preserving sparse codes. In *NeurIPS*, volume 35, pages 2320–2332, 2022.

[48] D. Yin, W. L. Tam, M. Ding, and J. Tang. Mrt: Tracing the evolution of scientific publications. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):711–724, 2021.

[49] Q. Zhang, S. Xu, Q. Chen, G. Sui, J. Xie, Z. Cai, Y. Chen, Y. He, Y. Yang, F. Yang, M. Yang, and L. Zhou. VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *OSDI*, pages 377–395, 2023.

[50] 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*, 16(8):1979–1991, 2023.

[51] B. Zheng, Z. Xi, L. Weng, N. Hung, H. Liu, and C. Jensen. Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search. *Proceedings of the VLDB Endowment*, 13(5):643–655, 2020.

APPENDIX

A. Suitability Analysis of Hausdorff Distance

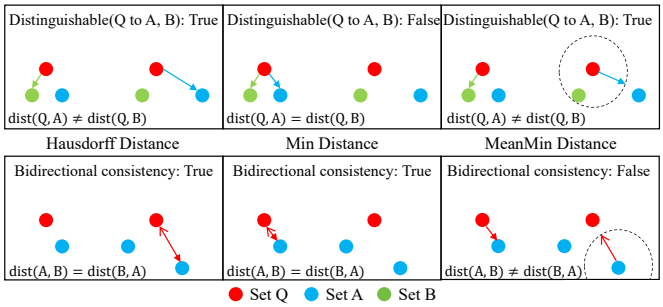


Figure 11: Comparative Analysis of Different Distance Measures

To demonstrate the advantages of the Hausdorff distance in vector set comparison, a comparative analysis of three distance measures is conducted: minimum distance, mean minimum distance, and Hausdorff distance. The focus is on illustrating the superior precision and symmetry of the Hausdorff distance.

Three vector sets $\mathbf{Q}$, $\mathbf{A}$, and $\mathbf{B}$ are used for the precision analysis, each containing two vectors. For the symmetry analysis, two vector sets $\mathbf{Q}$ and $\mathbf{A}$ are employed, where $\mathbf{Q}$ contains two vectors and $\mathbf{A}$ contains three vectors. For visualization purposes, all vectors are represented as points in a two-dimensional space. This representation allows for a clear illustration of the distance measures' properties.

The distance measures are defined as follows:

1) Minimum (Min) distance:

$$d_{min}(\mathbf{A}, \mathbf{B}) = \min_{\mathbf{a}\in\mathbf{A}, \mathbf{b}\in\mathbf{B}} d(\mathbf{a}, \mathbf{b}),$$

2) Mean Minimum (MeanMin) distance:

$$d_{mean-min}(\mathbf{A}, \mathbf{B}) = \frac{1}{|\mathbf{A}|} \sum_{\mathbf{a}\in\mathbf{A}} \min_{\mathbf{b}\in\mathbf{B}} d(\mathbf{a}, \mathbf{b}),$$

3) Hausdorff distance: As defined in Definition 4.

Here, $d(\cdot, \cdot)$ denotes the Euclidean distance, and $|\mathbf{A}|$ denotes the cardinality of vector set $\mathbf{A}$.

To evaluate precision, the distances between $\mathbf{Q}$ and vector sets $\mathbf{A}$ and $\mathbf{B}$ are analyzed, as shown in the first row of Figure 11. The specific distance matrices are presented below:

$\mathbf{Q}$ to $\mathbf{A}$	$\mathbf{Q}_1$	$\mathbf{Q}_2$
$\mathbf{A}_1$	1	5
$\mathbf{A}_2$	6	3

$\mathbf{Q}$ to $\mathbf{B}$	$\mathbf{Q}_1$	$\mathbf{Q}_2$
$\mathbf{B}_1$	1	5
$\mathbf{B}_2$	4	1

The analysis yields the following results:

1) $d_{min}(\mathbf{Q}, \mathbf{A}) = d_{min}(\mathbf{Q}, \mathbf{B}) = 1$
2) $d_{mean-min}(\mathbf{Q}, \mathbf{A}) = 2$, $d_{mean-min}(\mathbf{Q}, \mathbf{B}) = 1$
3) $d_H(\mathbf{Q}, \mathbf{A}) = 3$, $d_H(\mathbf{Q}, \mathbf{B}) = 2$

These results clearly demonstrate the superior precision of the Hausdorff distance. The minimum distance fails to differentiate between $\mathbf{Q}$'s relationships with $\mathbf{A}$ and $\mathbf{B}$. The mean minimum distance and Hausdorff distance clearly distinguish both the similarities and differences between the vector sets. While both $d_{mean-min}$ and d_H show discriminative power, we further examine their symmetry properties.

To examine symmetry, a case with two vector sets, $\mathbf{Q}$ and $\mathbf{A}$, where $\mathbf{Q}$ contains two vectors and $\mathbf{A}$ contains three vectors, is analyzed. The distance matrix is:

$\mathbf{Q}$ to $\mathbf{A}$	$\mathbf{Q}_1$	$\mathbf{Q}_2$
$\mathbf{A}_1$	1	4
$\mathbf{A}_2$	4	1
$\mathbf{A}_3$	7	3

The analysis yields the following results:

1) $d_{min}(\mathbf{Q}, \mathbf{A}) = d_{min}(\mathbf{A}, \mathbf{Q}) = 1$
2) $d_{mean-min}(\mathbf{Q}, \mathbf{A}) = 1$, $d_{mean-min}(\mathbf{A}, \mathbf{Q}) = 1.67$
3) $d_H(\mathbf{Q}, \mathbf{A}) = d_H(\mathbf{A}, \mathbf{Q}) = 3$

These results highlight the perfect symmetry of the Hausdorff distance. The minimum distance also exhibits symmetry. However, the mean minimum distance produces different results depending on the direction of comparison. The Hausdorff distance maintains consistency regardless of the order of vector set comparison, ensuring reliable and consistent similarity assessments.

The examples demonstrate the superiority of the Hausdorff distance in vector set comparisons. Its advantages are twofold. First, it provides enhanced precision in set differentiation. Second, it maintains consistent symmetry regardless of comparison direction. These properties are not present in other common measures. The Hausdorff distance effectively captures fine-grained differences while ensuring mathematical consistency. This balance makes it particularly suitable for complex vector set comparisons. As a result, it serves as a versatile and reliable metric across various applications.

B. Supplementary Experiments

TABLE X: Impact of Different Embedding Methods

Dataset	Embedding Model	Embedding Dimension	Recall (%) Top-3	Recall (%) Top-5	Time (s)
CS (1024)	MiniLM	384	98.9	98.2	0.42
CS (1024)	DistilUse	512	92.3	90.8	0.46
Picture (1024)	ResNet18	512	97.9	96.6	0.42
CS (2048)	MiniLM	384	98.4	97.3	0.43
CS (2048)	DistilUse	512	85.2	81.0	0.45
Picture (2048)	ResNet18	512	97.0	94.0	0.43

This section presents additional experiments and analyses further to validate the performance and versatility of `BioVSS`. We explore five key aspects: the impact of embedding models on vector representations, the effect of top-k on result quality, storage analysis on Medicine and Picture datasets, exploration of alternative distance metrics, and performance analysis via `BioHash` iteration.

1) Impact of Embedding Models on Algorithm Performance: to assess the robustness of `BioVSS` across various vector representations, we conducted experiments using different embedding models. These models inherently produce vectors of varying dimensions, allowing us to evaluate the method's performance across different vector lengths.

The experiments were designed to evaluate two critical aspects of embedding model impact. First, the performance consistency was tested using different embedding models within the same modality but with varying vector dimensions.

TABLE XI: Recall Rate for Top-k Across Different Datasets

Method	Top-3	Top-5	Top-10	Top-15	Top-20	Top-25	Top-30
CS Dataset							
BioVSS	98.6%	98.7%	98.5%	98.5%	98.4%	98.2%	98.1%
BioVSS++	98.9%	98.2%	97.5%	97.2%	96.9%	96.7%	96.4%
Medicine Dataset							
BioVSS	99.1%	98.8%	97.8%	97.5%	97.2%	97.0%	96.9%
BioVSS++	97.4%	96.0%	94.8%	93.9%	93.2%	92.7%	92.3%
Picture Dataset							
BioVSS	100%	99.9%	99.8%	99.7%	99.6%	99.6%	99.5%
BioVSS++	99.9%	99.9%	90.0%	80.0%	85.0%	80.0%	73.3%

Second, the method's versatility was examined across different modalities while maintaining consistent vector dimensions. CS and Picture datasets represent text and image modalities respectively, enabling these comparative analyses.

To validate the impact of different vector dimensions within the same modality, two text embedding models were applied to CS dataset. All-MiniLM-L6-v2[1] and distiluse-base-multilingual-cased-v2[1] from Hugging Face generate 384-dimensional and 512-dimensional vectors. Table X elucidates the efficacy and robustness of BioVSS across diverse embedding dimensionalities. For CS dataset with a Bloom filter cardinality of 1024, both the 384-dimensional MiniLM and 512-dimensional DistilUse models exhibit exceptional recall performance. MiniLM achieves Top-3 and Top-5 recall rates of 98.9% and 98.2%, while DistilUse demonstrates 92.3% and 90.8%. Significantly, the computational latencies are nearly equivalent irrespective of vector dimensionality, corroborating the dimension-agnostic nature of the search algorithm. The algorithm maintains similarly high recall rates when increasing the Bloom filter size to 2048. Notably, the search process exhibits dimension-invariant computational complexity, ensuring efficient performance across varying dimensional scales.

To validate the impact of different modalities with the same vector dimension, embedding models generating 512-dimensional vectors were applied to CS and Picture datasets. Specifically, with a Bloom filter cardinality of 1024, both CS and Picture datasets demonstrate high recall rates (90% for Top-5), with comparable computational latencies (0.4s+). This performance is maintained when scaling to a size of 2048, evidencing the algorithm's robustness across data modalities.

These experiments validate that BioVSS's performance remains stable across varying embedding dimensions and data modalities.

2) Impact of Top-k on Result Quality: to evaluate the impact of the top-k parameter on result quality, we conducted experiments using default parameters for both BioVSS and BioVSS++ on CS, Medicine, and Picture datasets. Table XI presents the recall rates for various top-k values ranging from 3 to 30. On CS dataset, BioVSS demonstrates consistent performance, maintaining a high recall rate above 98% across all top-k values. BioVSS++ shows slightly higher recall for top-3 (98.9%) but experiences a gradual decrease as k increases, reaching 96.4% for top-30. Medicine dataset reveals a similar trend, with BioVSS maintaining high recall rates (99.1% for top-3, decreasing slightly to 96.9% for top-30), while BioVSS++ shows a decline (from 97.5% for top-3 to 92.3% for top-30). Picture dataset exhibits a similar trend. The slight performance degradation observed in BioVSS++ for larger k values suggests a trade-off between efficiency and recall, which may be attributed to its more aggressive filtering mechanism. Notably, both methods maintain high recall rates (above 92%) even for larger k values, demonstrating their effectiveness in retrieving relevant results across various retrieval scenarios. The filtering mechanism of BioVSS++ has brought significant improvements in query efficiency. Smaller top-k values are typically more valuable and often yield the most relevant results in many applications. Consequently, we will focus on BioVSS++ in our subsequent experiments.

3) Storage Analysis on Medicine and Picture Datasets: Tables XII and XIII present the storage efficiency analysis on Medicine and Picture datasets. These results corroborate the findings from CS, demonstrating consistent storage reduction patterns across different data domains.

TABLE XII: Filter Storage Comparison on Medicine Dataset

Bloom	L	Count Bloom Space (GB)			Binary Bloom Space (GB)		
		Dense	COO	CSR	Dense	COO	CSR
1024	16	7.5	0.57	0.29	0.94	0.19	0.1
	32		1.13	0.57		0.38	0.19
	48		1.67	0.84		0.56	0.28
	64		2.18	1.1		0.73	0.37
2048	16	15	0.61	0.31	1.87	0.2	0.11
	32		1.18	0.6		0.39	0.2
	48		1.73	0.87		0.58	0.29
	64		2.29	1.15		0.76	0.38

TABLE XIII: Filter Storage Comparison on Picture Dataset

Bloom	L	Count Bloom Space (GB)			Binary Bloom Space (GB)		
		Dense	COO	CSR	Dense	COO	CSR
1024	16	20.55	2.79	1.41	2.57	0.93	0.48
	32		5.14	2.58		1.71	0.87
	48		7.26	3.64		2.42	1.22
	64		9.22	4.62		3.07	1.55
2048	16	41.1	3.01	1.51	5.14	1	0.51
	32		5.6	2.81		1.87	0.94
	48		8.02	4.02		2.67	1.35
	64		10.71	5.37		3.57	1.8

4) Exploration of Alternative Distance Metrics: while the main text demonstrates the effectiveness of BioVSS++ using Hausdorff distance, the framework's applicability extends to other set-based distance metrics. To further validate this extensibility, additional experiments were conducted using alternative distance measures.

Our experiments compared BioVSS++ against DESSERT[3] under various parameter configurations (Table XIV). Using the MeanMin distance metric, BioVSS++ achieved a Top-3 recall of 59.0% with a query time of 0.46 seconds. These results demonstrate reasonable efficiency beyond the Hausdorff setting. The performance disparity between Hausdorff and MeanMin distances arises from their aggregation mechanisms. Hausdorff utilizes a three-level structure (*min- max- max*), whereas MeanMin employs a two-level aggregation (*min-mean*). Such three-level aggregation creates more distinctive distance distributions. Consequently, similar sets exhibit higher

[3]https://github.com/ThirdAIResearch/Dessert. Without IVF indexing.

TABLE XIV: Performance Comparison of MeanMin

Method	Top-3	Top-5	Time
DESSERT (tables=32, hashes_per_t=6)	45.9%	35.8%	0.21s
DESSERT (tables=32, hashes_per_t=12)	40.3%	28.8%	0.56s
DESSERT (tables=24, hashes_per_t=6)	42.3%	32.6%	0.18s
DESSERT (num_t=24, hashes_per_t=12)	38.7%	27.6%	0.36s
BioVSS++ (default_parameter)	59.0%	51.6%	0.46s

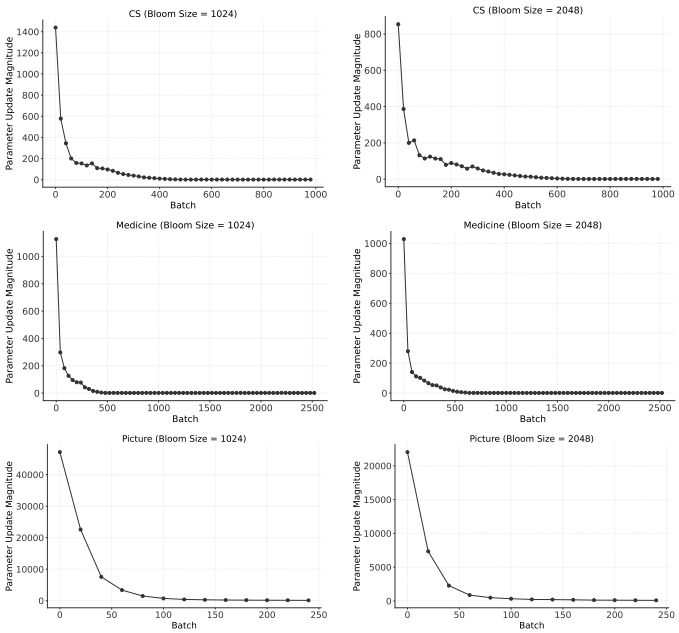


Figure 12: Parameter Update Magnitude Across Batches in BioHash

collision probabilities, while dissimilar sets are more effectively separated.

The extensibility to different metrics stems from BioVSS++'s decoupled design. The filter structure operates independently of the specific distance metric. This architectural choice enables metric flexibility while maintaining the core filtering mechanisms.

5) Performance Analysis via BioHash Iteration: BioHash algorithm serves as a crucial component in our framework, with its reliability directly impacting system performance. While BioHash has been validated in previous research [37], a rigorous examination of its behavior within our specific application domain is necessary.

The iteration count parameter plays a vital role in BioHash's computational process, specifically in determining learning efficacy. At its core, BioHash implements a normalized gradient descent optimization approach. The magnitude of parameter updates functions as a key metric for quantifying learning dynamics. This magnitude is defined as:

$$M_t = \max_{i,j} |\Delta W_{ij}^t|,$$

where M_t represents the update magnitude at iteration t, and ΔW_{ij}^t denotes the weight change for the synaptic connection between neurons i and j. This metric effectively captures the most substantial parametric modifications occurring within the network during each training iteration ($batch\_size = 10k$).

Figure 12 illustrates the parameter update magnitude dynamics across three distinct datasets (CS, Medicine, and Picture) under varying Bloom filter sizes (1024 and 2048 bits). The experimental results reveal several significant patterns:

- **Convergence Behavior:** Across all configurations, the parameter update magnitude exhibits a consistent decay pattern. The parameter updates show high magnitudes during initial batches, followed by a rapid decrease and eventual stabilization. This pattern indicates BioHash algorithm's robust convergence properties regardless of the domain-specific data characteristics.
- **Filter Size Consistency:** The comparison between 1024 and 2048 Bloom filter configurations demonstrates remarkable consistency in convergence patterns. This observation suggests that BioHash maintains stable performance characteristics independent of filter size, validating the robustness of our parameter update mechanism across different capacity settings.
- **Cross-Domain Consistency:** The similar convergence patterns observed across CS, Medicine, and Picture datasets validate the algorithm's domain-agnostic nature. Despite the inherent differences in data distributions, BioHash iteration mechanism maintains consistent performance characteristics.

Through extensive empirical analysis across diverse application scenarios, we validated BioHash's effectiveness as a core component of our system. Our experiments demonstrated consistent and reliable convergence behavior under various environmental configurations. The results confirmed that BioHash achieves stable parameter updates well before completing the full training process.

6) Query Time Analysis: experiments evaluated query efficiency across candidate sets ranging from 20k to 50k. As illustrated in Table XV, query time scales approximately linearly with candidate count. With the configuration of WTA=64 and Bloom filter size=1024, the method achieves consistent and efficient query times (0.44s-0.51s for 50k candidates) across all datasets. While reducing WTA hash count to 16 decreases query time by up to 15%, such reduction compromises recall performance as shown in Figures 6 and 7. Additionally, doubling the Bloom filter size to 2048 offers minimal efficiency gains ($\leq 0.03s$ improvement), making the added memory overhead unjustifiable. These observations support selecting 64 and 1024 as the optimal configurations balancing efficiency and effectiveness.

C. Theoretical Analysis of Dual-Layer Filtering Mechanism

BioFilter framework demonstrates potential compatibility with diverse set-based distance metrics. This versatility stems from its filter structure being independent of specific distance measurements. This dual-layer approach achieves efficiency through progressive refinement: the count Bloom filter-based inverted index rapidly reduces the search space, while the binary Bloom filter-based sketches enable similarity assessment of the remaining candidates. To establish the theoretical foundation, we introduce the concept of set connectivity and analyze its relationship with filter collision patterns.

TABLE XV: Query Time (s) with Different Bloom Filter and WTA for Top-3&5

Dataset	Bloom = 1024, WTA=64				Bloom = 1024, WTA=48				Bloom = 1024, WTA=32				Bloom = 1024, WTA=16			
	50k	40k	30k	20k	50k	40k	30k	20k	50k	40k	30k	20k	50k	40k	30k	20k
CS	0.44	0.35	0.29	0.21	0.46	0.48	0.38	0.29	0.44	0.46	0.31	0.28	0.43	0.45	0.37	0.27
Medicine	0.51	0.40	0.34	0.24	0.49	0.41	0.32	0.25	0.47	0.39	0.28	0.22	0.45	0.36	0.28	0.19
Picture	0.44	0.36	0.28	0.20	0.45	0.36	0.29	0.20	0.45	0.39	0.30	0.21	0.45	0.38	0.29	0.21
Dataset	Bloom = 2048, WTA=64				Bloom = 2048, WTA=48				Bloom = 2048, WTA=32				Bloom = 2048, WTA=16			
	50k	40k	30k	20k	50k	40k	30k	20k	50k	40k	30k	20k	50k	40k	30k	20k
CS	0.45	0.34	0.28	0.20	0.44	0.33	0.27	0.20	0.43	0.33	0.27	0.20	0.39	0.32	0.26	0.19
Medicine	0.48	0.40	0.31	0.22	0.47	0.39	0.29	0.22	0.46	0.38	0.29	0.21	0.43	0.37	0.28	0.19
Picture	0.43	0.35	0.27	0.20	0.46	0.37	0.29	0.20	0.43	0.37	0.28	0.20	0.43	0.37	0.28	0.20

Definition 11 (**Set Connectivity**). *For two vector sets* $\mathbf{Q}$ *and* $\mathbf{V}$*, their set connectivity is defined as:*

$$Conn(\mathbf{Q}, \mathbf{V}) = \sum_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{v}\in\mathbf{V}} sim(\mathbf{q}, \mathbf{v}),$$

where $sim(\mathbf{q}, \mathbf{v})$ *represents a pairwise similarity.*

A fundamental insight of our framework is that the effectiveness of both Bloom filters stems from their ability to capture set relationships through hash collision positions. Specifically, when vector sets share similar elements, their hash functions map to overlapping positions, manifesting as either accumulated counts in the count Bloom filter or shared bit patterns in the binary Bloom filter. This position-based collision mechanism forms the theoretical foundation for both filtering layers. The following theorem establishes that such collision patterns in both filter types correlate with set connectivity, thereby validating the effectiveness of our dual-layer approach:

Theorem 5 (**Collision-Similarity Relationship**). *For a query vector set* $\mathbf{Q}$ *and two vector sets* $\mathbf{V_1}$ *and* $\mathbf{V_2}$*, where* $\mathbf{Q} \cap_h \mathbf{V}$ *denotes hash collisions between elements from* $\mathbf{Q}$ *and* $\mathbf{V}$ *in either count Bloom filter or binary Bloom filter. If their collision probability satisfies:*

$$P(\mathbf{Q} \cap_h \mathbf{V}_1 \neq \emptyset) \geq P(\mathbf{Q} \cap_h \mathbf{V}_2 \neq \emptyset),$$

Then:

$$Conn(\mathbf{Q}, \mathbf{V}_1) \gtrsim Conn(\mathbf{Q}, \mathbf{V}_2),$$

where $\gtrsim$ *denotes that the left-hand side is approximately greater than the right-hand side.*

Proof. By the definition of collision probability:

$$P(\mathbf{Q} \cap_h \mathbf{V}_1 \neq \emptyset) = 1 - \prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_1} (1 - sim(\mathbf{q}, \mathbf{v})),$$

$$P(\mathbf{Q} \cap_h \mathbf{V}_2 \neq \emptyset) = 1 - \prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_2} (1 - sim(\mathbf{q}, \mathbf{v})).$$

The given condition implies:

$$1 - \prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_1} (1 - sim(\mathbf{q}, \mathbf{v})) \geq 1 - \prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_2} (1 - sim(\mathbf{q}, \mathbf{v})).$$

Therefore:

$$\prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_1} (1 - sim(\mathbf{q}, \mathbf{v})) \leq \prod_{\mathbf{q}\in\mathbf{Q}} \prod_{\mathbf{v}\in\mathbf{V}_2} (1 - sim(\mathbf{q}, \mathbf{v})).$$

Taking the logarithm of both sides:

$$\sum_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{v}\in\mathbf{V}_1} \log(1 - sim(\mathbf{q}, \mathbf{v})) \leq \sum_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{v}\in\mathbf{V}_2} \log(1 - sim(\mathbf{q}, \mathbf{v})).$$

Using the Taylor series expansion of $\log(1 - x)$ at $x = 0$:

$$\log(1 - x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \ldots, \quad x \in [0, 1]$$

Approximating this for small x:

$$\log(1 - x) \approx -x + O(x^2).$$

Since $sim(\mathbf{q}, \mathbf{v}) \in [0, 1]$ is a small value, we can approximate the following relationship:

$$\sum_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{v}\in\mathbf{V}_1} sim(\mathbf{q}, \mathbf{v}) \gtrsim \sum_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{v}\in\mathbf{V}_2} sim(\mathbf{q}, \mathbf{v})$$

This establishes:

$$Conn(\mathbf{Q}, \mathbf{V}_1) \gtrsim Conn(\mathbf{Q}, \mathbf{V}_2)$$

□

This theoretical analysis demonstrates that higher collision probability in our filters correlates with stronger set connectivity, providing a foundation for the effectiveness of `BioFilter`. The relationship between hash collisions and set connectivity validates that our dual-layer filtering mechanism effectively preserves and identifies meaningful set relationships during the search process, while the metric-independent nature of this correlation supports the framework's potential adaptability to various set distance measures.