LSH-Chamfer Trees: Efficient Approximation of Multi-Vector Similarity

Luigi Liu
Department of Computer Science
Columbia University
ll3840@columbia.edu

Yuval Shemla
Department of Computer Science
Columbia University
ys3571@columbia.edu

Keemin Lee
Department of Computer Science
Columbia University
kjl2175@columbia.edu

Alex Racapé
Department of Computer Science
Columbia University
abr2184@columbia.edu

May 2025

Abstract

Multi-vector late-interaction models such as ColBERT deliver state-of-the-art retrieval accuracy, but their *Chamfer* (MaxSim) scoring makes query time grow linearly with the total number of token embeddings. We re-cast Chamfer search as an approximate nearest-neighbour problem in an *LSH-Forest*. Our analysis shows that, with $T = \Theta!\big(n^{\rho}\big)$ trees and $L = \Theta!\big(\log(m/\delta)\big)$ independent forests, the proposed *LSH-Forest Chamfer* algorithm returns a $(1-\varepsilon)$-approximate top document with probability at least $1-\delta$ and runs in sub-linear $\mathcal{O}\big(m, n^{\rho}\big)$ time while using $\mathcal{O}!\big(n^{1+\rho}\big)$ memory, where the exponent $\rho < 1$ depends only on ε. We further introduce a depth-weighted voting heuristic that exploits the exponential decay of hash-collision probability to approximate Chamfer without max-pooling, reducing query time to $\mathcal{O}\big(|Q|, T, \log(N\bar{P})\big)$. Experiments on simulated data and the HotpotQA benchmark confirm that our method achieves up to 90% Recall@100 with two orders-of-magnitude fewer inner products than exact Chamfer, and highlight the importance of tree balancing and dimensionality reduction for high-dimensional embeddings. Taken together, these results demonstrate that depth-aware LSH forests provide a principled, scalable alternative to existing multi-vector retrieval engines while retaining strong theoretical guarantees.

1 Introduction

Over the past several years, multi-vector representations have emerged as a transformative paradigm in information retrieval. Traditional methods such as Biencoders, generate a single neural embedding for each document in the dataset[1]. However, these models fail to generalize well to new data sets since

[1]https://arxiv.org/pdf/2408.01094v1

it is difficult to learn a single document embedding that can capture the relevant information for all possible queries. Frameworks like ColBERT introduced late interaction as an innovative mechanism for estimating similarity between queries and documents[2]. In this approach, the model generates sets of vectors for each document, enabling more nuanced representation and comparison. Typically, there is an embedding for each token, drastically increasing the dimensionality of the problem. With late interaction, the similarity between the best documents are found by computing the similarity between each query and all document representations. This introduces a critical tension between representation richness and computational efficiency.

The approach hinges on efficiently computing similarity between query and document representations, with Chamfer similarity (MaxSim) emerging as the standard metric. However, this method introduces significant computational challenges, and existing approaches such as MUVERA and PLAID have attempted to mitigate these limitations through advanced indexing, compression, and locality-sensitive hashing (LSH) strategies[3]. Despite these innovations, a fundamental gap remains in developing a retrieval method that can efficiently scale with increasing vector complexity while maintaining high accuracy across diverse datasets.

In this work, we introduce a novel retrieval method that approximates Chamfer similarity with LSH forests. By reframing the problem of multi-vector retrieval through the lens of locality-sensitive hashing and candidate vector generation, we introduce a fundamentally different approach to approximating similarity efficiently. Specifically, our contributions include:

1. A new retrieval method that provides a strong approximation of Chamfer similarity.
2. Theoretical analysis providing provable guarantees on the approximation quality of our retrieval mechanism.
3. A comprehensive empirical evaluation across simulated and real-world datasets, demonstrating the method's robustness and performance.
4. Extensions of our framework and avenues for future research.

The subsequent sections of this paper are organized to systematically present our approach. We begin with a review of related work in Section 2. Section 3 explores the theoretical bounds and rationale behind our approach, Section 4 details how it was built in practice. Section 5 evaluates the method on simulated data and real-world datasets, and Section 6 introduces a novel heuristic that further approximates the Chamfer distance for multi-vector retrieval.

[2]https://arxiv.org/pdf/2004.12832
[3]https://arxiv.org/pdf/2405.19504 and https://arxiv.org/pdf/2205.09707

2 Related Work

2.1 Multi-vector Retrieval

Late interaction models like ColBERT pioneered the use of multi-vector representations for retrieval, encoding each query and document as a set of token embeddings and scoring their similarity via the Chamfer similarity (also known as MaxSim). Formally, given a query vector set Q and document vector set D, Chamfer similarity is defined as

$$\text{Chamfer}(Q, D) = \sum_{q \in Q} \max_{d \in D} \langle q, , d \rangle$$

This is simply summing the maximum inner product for each query embedding. ColBERT demonstrated that this token-level matching significantly improves ranking effectiveness over single-vector representations, but at the cost of far more embeddings and expensive search operations. Subsequent systems have aimed to accelerate multi-vector retrieval. ColBERTv2 introduced a multi-stage retrieval pipeline where document token embeddings are first clustered via k-means and initial matching is done against cluster centroids, then further stages progressively refine the candidates. The PLAID engine builds on this approach with an optimized four-stage filtering and pruning process that dramatically reduces search time.

More recently, MUVERA (Multi-Vector Retrieval via Fixed Dimensional Encodings) took a different approach by compressing each document's embedding set into a single fixed-length vector representation or FDE. MUVERA computes these FDE vectors such that a single inner product between a query's FDE and a document's FDE approximates their Chamfer similarity. This yields a single-vector proxy for multi-vector search, enabling the use of standard MIPS indices while preserving accuracy. The authors show that MUVERA's FDE-based retrieval achieves strong recall of true Chamfer nearest neighbors, with theoretical guarantees on approximation quality. Empirically, it matches or surpasses PLAID's retrieval quality with far lower latency. One limitation of this work is that the FDE size does not scale well with document length. MUVERA assumes that documents consist of a limited number of embeddings, and they selected datasets with only 18-165 average tokens per document. They derive a $1 + \varepsilon$ approximation of Chamfer similarity with $1 - \delta$ probability, resulting in the following bound where $m = |Q| + |P|$:

$$d_{\text{FDE}} = \left(\frac{m}{\delta}\right)^{\mathcal{O}\left(\frac{1}{\varepsilon}\right)}$$

In other words, the dimension of the FDE must be extremely large in order to maintain their guarantees, especially when there are many vectors per document. Our method diverges from these prior efforts by proposing a new approximation method that does not depend on querying every document vector or fixing the number of vectors per document. Instead of token-level search or static compression, we employ a specialized indexing strategy based on LSH forests to directly approximate the Chamfer similarity for each document.

2.2 LSH Forests

LSH Forest is a data-dependent indexing scheme for similarity search that offers adaptability and efficiency through self-tuning hash structures. Originally introduced by Bawa et al. (2005), LSH Forest organizes data points into multiple prefix trees (tries) built on random hash functions. An LSH Forest grows each tree path until the indexed point is sufficiently distinguished from others, assigning each point a variable-length hash key. This adaptive depth effectively tunes the tree to the data distribution: dense regions of the space yield longer branches while sparser areas are shallower, ensuring a balanced partition without manual parameter tuning. The original work demonstrated LSH Forest's self-tuning nature and superior empirical performance on similarity search tasks. In follow-up work, Andoni et al. (2017) provided a theoretical foundation for LSH Forest by showing that a modified algorithm can provably outperform standard data-oblivious LSH in the worst case. Their modification introduced additional pivot points per node which improve the query time exponent compared to vanilla LSH. Building on this foundation, we apply LSH Forest in a new context: approximating Chamfer similarity in multi-vector IR. We adapt the LSH Forest approach to index documents' token embeddings such that the nearest matches for each query embedding can be retrieved efficiently within the forest. In essence, the LSH Forest's adaptive hashing allows our index to focus on the most relevant vector matches for a query, aggregating those to approximate the Chamfer score.

2.3 Chamfer Approximation

Bakshi et al. introduce the first $(1 + \epsilon)$–approximation algorithm whose running time is near-linear in the total number of points for the classical Chamfer distance between two finite point sets. The core routine combines two ingredients. First they obtain crude nearest neighbor estimates using a multi-level LSH structure. Then they use importance sampling over the retrieved candidates and calculate the exact nearest-neighbor distance. Overall, this method achieves a runtime of $O(nd \log(n/\varepsilon^2)$, which drastically improves upon Chamfer's naive $O(n^2)$ runtime. In our setting, the naive approach would involve calculating the Chamfer distance between the queries and every set of document vectors. This approximation method improves the runtime, but it would still depend on the number of vectors per document. Meanwhile, our LSH forest removes this dependance.

3 Theory

In this section we formalise the *LSH-Forest Chamfer* algorithm, prove a $(1-\varepsilon)$-multiplicative guarantee on the returned Chamfer similarity, and derive bounds on its expectation, variance and query time.

3.1 Problem formulation

Data model. Let $\mathcal{D} = \{P_1, \dots, P_n\}$ be a corpus in which every document is a multiset of unit[4] vectors, $P_i = \{p_{i,1}, \dots, p_{i,|P_i|}\} \subset \mathbb{S}^{d-1}$. A query is a multiset $Q = \{q_1, \dots, q_m\} \subset \mathbb{S}^{d-1}$.

Chamfer (MaxSim) similarity. For a token q_j and document P_i define

$$M_j(P_i) \ := \ \max_{p \in P_i} \langle q_j, p \rangle. \tag{1}$$

The *Chamfer* (a.k.a. MaxSim) score is

$$\mathrm{Chamfer}(Q, P_i) \ := \ \sum_{j=1}^{m} M_j(P_i). \tag{2}$$

Goal. Given $(Q, \mathcal{D})$, return a document $P_{\mathrm{ret}} \in \mathcal{D}$ satisfying

$$\mathrm{Chamfer}(Q, P_{\mathrm{ret}}) \ \geq \ (1-\varepsilon) \max_i \mathrm{Chamfer}(Q, P_i) \quad \text{with probability } 1-\delta. \tag{3}$$

3.2 Per-token success of an LSH-Forest

Let $\mathcal{H}$ be an $(\varepsilon, f(\varepsilon))$-*sensitive* hash family in the sense of [4, Def. 2]. Theorem 5.1 of that paper states:

> **Theorem 5.1 ([4]).** With $\ell = n^{f(\varepsilon)}$ independent prefix tries, an *LSH-Forest* returns an ε-approximate nearest neighbour with probability at least a constant $c_0 \in (0,1)$ that is independent of n, d and of the query.

Hence for any token q_j and any document P_i, if $\tilde{p}_j$ is the point returned by one forest we have

$$\Pr\Big[\langle q_j, \tilde{p}_j \rangle \geq (1-\varepsilon) M_j(P_i) \Big] \ \geq \ p_0, \qquad p_0 := c_0. \tag{4}$$

(The cosine form follows from $\|q-p\|_2^2 = 2\big(1 - \langle q, p \rangle\big)$ for unit vectors.)

[4]All vectors are ℓ_2-normalised at indexing time, so cosine similarity equals the inner product.

3.3 Boosting with L independent forests

Build L independent forests. For a fixed q_j the events across forests are independent, so

$$\Pr[\text{token } q_j \text{ fails in all } L \text{ forests}] \;=\; (1-p_0)^L \;\le\; e^{-p_0 L}.$$

Choose

$$L \;:=\; \left\lceil \tfrac{\log(m/\delta)}{p_0} \right\rceil. \tag{5}$$

Then

$$\Pr[\text{token } q_j \text{ fails}] \;\le\; \frac{\delta}{m}. \tag{6}$$

3.4 Success for all tokens

By the union bound over the m tokens,

$$\Pr[\exists\, j\colon \text{ token } q_j \text{ fails}] \;\le\; \sum_{j=1}^{m} \frac{\delta}{m} \;=\; \delta. \tag{7}$$

Thus, with probability at least $1-\delta$ every token finds a $(1-\varepsilon)$-approximate match in *some* forest.

3.5 Chamfer approximation

Condition on the event that all tokens succeed. Let $\tilde{p}_j$ denote the best candidate for q_j among all L forests. By (4), $\langle q_j, \tilde{p}_j\rangle \ge (1-\varepsilon)M_j(P_i)$ for every j. Summing and using (2),

$$\widetilde{C}_i := \sum_{j=1}^{m} \langle q_j, \tilde{p}_j\rangle \;\ge\; (1-\varepsilon)\,\mathrm{Chamfer}(Q, P_i). \tag{8}$$

Return $P_{\mathrm{ret}} = \arg\max_i \widetilde{C}_i$. For the optimal document $P^\star$,

$$\mathrm{Chamfer}(Q, P_{\mathrm{ret}}) \;\ge\; \widetilde{C}_{P^\star} \;\ge\; (1-\varepsilon)\,\mathrm{Chamfer}(Q, P^\star).$$

Combining with (7) yields the claimed $(1-\varepsilon)$-approximation with probability $1-\delta$.

3.6 Expectation and variance of the estimator

For each token q_j define the indicator $X_j = \mathbf{1}\{\langle q_j, \tilde{p}_j\rangle \ge (1-\varepsilon)M_j(P_i)\}$. Independence across tokens is *not* required below; we only use linearity of expectation and the fact that $0 \le X_j \le 1$.

Expectation. Using (8) and $\mathbb{E}[X_j] = p_0$,

$$\mathbb{E}[\widetilde{C}_i] \;=\; (1-\varepsilon)p_0 \sum_j M_j(P_i) \;=\; (1-\varepsilon)p_0\ \mathrm{Chamfer}(Q, P_i).$$

Variance. Since $X_j^2 = X_j$ and $M_j(P_i) \le 1$,

$$\mathrm{Var}[\widetilde{C}_i] \;=\; \sum_j \mathrm{Var}\big[(1-\varepsilon)X_j M_j(P_i)\big] \;\le\; m(1-\varepsilon)^2 p_0(1-p_0).$$

Chebyshev therefore gives $\Pr\big[\widetilde{C}_i < (1-\varepsilon)p_0\ \mathrm{Chamfer}(Q,P_i) - \tau\big] \le \mathrm{Var}[\widetilde{C}_i]/\tau^2$, and a Chernoff bound yields exponential tails.

Why the scaling factors appear.[5] The random estimator $\widetilde{C}_i$ is built from approximate neighbours, so every successful match is *already* down-scaled by the approximation factor $(1-\varepsilon)$. In addition, a token succeeds in at least one of the L forests only with probability p_0; with probability $1-p_0$ it contributes nothing (or a negligible value). Multiplying these two effects yields the prefactor $(1-\varepsilon)p_0$ in the expectation.

Variance bound. Because each indicator X_j is Bernoulli with mean p_0 and $X_j^2 = X_j$, the variance of every summand $(1-\varepsilon)X_j M_j(P_i)$ is at most $(1-\varepsilon)^2 p_0(1-p_0)\, M_j(P_i)^2 \le (1-\varepsilon)^2 p_0(1-p_0)$, using $M_j(P_i) \le 1$. Summing over the m independent[6] tokens gives the claimed upper bound $m(1-\varepsilon)^2 p_0(1-p_0)$. The Chebyshev and Chernoff tail inequalities then quantify how rarely $\widetilde{C}_i$ deviates downward from its mean by more than a chosen margin τ.

3.7 Query complexity

The pivoted version of LSH-Forest [2, Thm. 4.1] implies that a single tree succeeds with probability $n^{-\rho}$ where

$$\rho \;<\; \frac{1}{\ln 4\,(c-2)} + O\big(c^{-2}\big), \qquad c := \frac{1}{1-\varepsilon}. \tag{9}$$

Taking $O(n^\rho)$ trees makes $p_0 \ge 0.9$. Hence

- **Per token:** $O(n^\rho)$ bucket probes and $O(1)$ inner products.
- **Whole query:** $T(m,n) = O\big(m\,n^\rho\big)$.
- **Index size:** $O\big(n^{1+\rho}\big)$ vectors (each tree contains one copy).

Because $\rho < 1$, the query time is sub-linear in the corpus size n.

3.8 Main result

Theorem 1 (LSH-Forest Chamfer)**.** *Fix $\varepsilon, \delta \in (0,1)$ and set $L = \lceil \log(m/\delta)/p_0 \rceil$ according to* (5)*. With probability at least $1-\delta$ the algorithm returns*

$$\mathrm{Chamfer}(Q, P_{\mathrm{ret}}) \;\ge\; (1-\varepsilon)\max_i \mathrm{Chamfer}(Q, P_i).$$

[5]Readers sometimes expect the estimator's expectation to equal the true Chamfer score. Here it is *proportional* instead, for two separate reasons.

[6]We do not assume independence *between* tokens—we merely add upper bounds—so the result is valid even if the tokens are correlated through the query.

Moreover

$$\mathbb{E}[\widetilde{C}_i] = (1-\varepsilon)p_0 \text{ Chamfer}(Q, P_i), \qquad \mathrm{Var}[\widetilde{C}_i] \le m(1-\varepsilon)^2 p_0(1-p_0),$$

and the query runs in $O\big(m\, n^{\rho}\big)$ time using $O\big(n^{1+\rho}\big)$ memory, where ρ is given by (9)*.*

3.9 Conclusion

The LSH-Forest Chamfer algorithm achieves a provable $(1-\varepsilon)$ multiplicative approximation with probability $1-\delta$, has expectation and variance that scale linearly with the query length m, and enjoys sub-linear query time $O(m\, n^{\rho})$ in the corpus size. The analysis relies only on (i) the $(\varepsilon, f(\varepsilon))$-sensitive hash family of [4] and (ii) the pivoted-tree success exponent of [2]; all other steps use elementary probability.

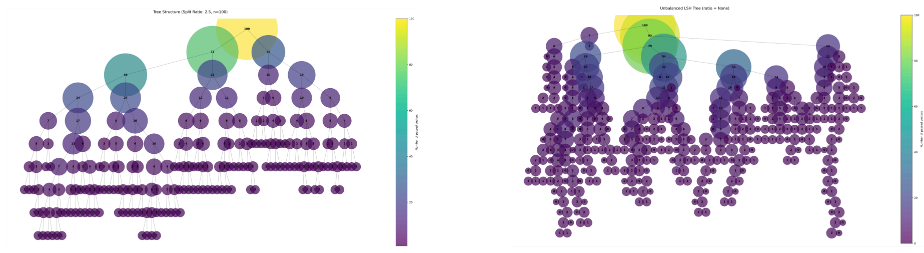
4 Implementation

Our LSH forest implementation follows the structure outlined by Bawa et al. in LSH Forest: Self-Tuning Indexes for Similarity Search. To approximate Chamfer, we instantiate a forest for each collection of document vectors. While the original paper demonstrates the design using Bit-sampling over $0, 1^d$, we use random projection onto a hyperplane (SimHash) so that the same forest can index real-valued transformer embeddings of dimension d. For each forest we instantiate l trees of max depth k_m, and each internal node caches a set of up to k descendant vectors. For querying, we follow their top-down and bottom-up approach to querying. For a given query vector, we descend each tree along matching hashes until a mismatch or depth k_m then synchronously climb upward, adding the cached descendants to a candidate set until we have at least a near neighbors. Finally, we rank the candidates according to the negative inner product.

Deviating from their implementation, we added a mechanism for dynamically balancing the tree during preprocessing. To achieve a more balanced tree, we added an option to retry hashing at each node until the split surpasses a balance threshold. Figure X, illustrates the structure of our tree with and without balancing with 100 vectors. Our design preserves the forest's sub-linear lookup and parameter-free tuning while decoupling search cost from the number of vectors per document. In the future, this query process could be made more efficient processing the multiple query vectors in parallel, or alternatively the trees could be partitioned across multiple servers if the scale is large enough.

While implementing this structure we noted that preprocessing and construction time was significant, and limited the scale of our experiments which necessitated repeatedly reconstructing the forests. However, our primary focus is on the query runtime which is $O(n||Q|(l \cdot k_m \cdot d + ad))$. There is a dependence on the number of queries and number of documents since we find an approximate neighbor from each document's forest for each query. Although, $O(n|Q|)$

is a theoretical improvement over brute force $O(n|Q| \max_i |P_i|)$, directly calculating the Chamfer similarity was faster on small datasets in practice since the majority of the calculation can be condensed into a single matrix multiplication.



Split Ratio = 2.5 Split Ratio = None

Figure 1: Tree Visualization with and without Balancing.

4.1 Experimental Setup

We designed our experiments with a focus on evaluating the effectiveness and efficiency of our LSH forest-based Chamfer approximation method. Our primary goals were to:

- Verify the functionality of our LSH forest.
- Compare our approximations to the true Chamfer similarity.
- Assess the accuracy of our LSH-based approximation in retrieving true Chamfer nearest neighbors in both simulated and real data settings.
- Measure the computational performance of our method, including query time and space complexity.

4.2 Hypothesis

We hypothesized that our LSH-based method would provide a strong approximation of Chamfer similarity while maintaining significantly lower computational overhead compared to a pure Chamfer distance-based approach. Specifically, we expected that:

- Our LSH-based approach would achieve at least 90% accuracy with established nearness thresholds while using significantly less memory and query time than a pure Chamfer approach.
- Preserve recall and query efficiency due to the adaptive nature of LSH forests and in the self-tuning case, the computational efficiency of scoring based on preprocessed weights as opposed to Chamfer distances.

4.3 Datasets

Our evaluation initially explored multiple BEIR benchmark datasets. However, we ultimately focused on HotpotQA, a multi-hop question answering dataset, due to its diversity in query complexity and larger usable document embedding count (446 vectors/document versus, for example, the 17 vectors/document of MS MARCO, another popular dataset we initially used) which was more similar to the simulated data parameters used (1000 vectors/document). We used the multi-qa-mpnet-base-dot-v1 model from the SentenceTransformers Python framework for semantic search embeddings. To further optimize computational efficiency and scalability, we applied PCA for dimensionality reduction, reducing the vector dimensionality of the HotpotQA embeddings to 25.

While the HotpotQA dataset contains far more than the 100 documents we used, time constraints pushed us toward this quick experimental setup. We also note that the documents used were the longest in word count in the dataset, allowing for a greater number of vectors per document. This ensured that our LSH-based method would be tested for efficiently handling high-dimensional real-world data without sacrificing significant accuracy.

In addition to real-world datasets, we also generated simulated data with, allowing us to systematically test the sensitivity of our method to noisy and high-dimensional data. For these datasets we used a vector dimension of 128 to match the output size of ColBERT, and we tested with 15 query vectors. By default we used 10 trees per forest, a max tree depth of 15, with 10 descendants cached per node. To verify the effectiveness of a single forest, we used 10,000 vectors. For the other experiments using simulated data, we evaluated 1,000 documents which each contained 100 vectors.

4.4 Evaluation Metrics

We evaluated our method using the following metrics:

- Recall@N: The proportion of queries for which the true top-N Chamfer nearest neighbors are correctly retrieved.

5 Results

5.1 Performance on Simulated Data

For the simulated dataset, we observed a consistent and expected trend: Recall@100 increased steadily with the number of trees (1). As the forest grew, the approximate nearest neighbors retrieved by the LSH forest more closely matched the true nearest neighbors calculated using Euclidean distance. This is consistent with theoretical expectations — increasing the number of trees in an LSH forest increases the probability of capturing true neighbors across the hash partitions. In the simulated scenario, Recall@100 eventually approached 1.0, indicating near-perfect retrieval.

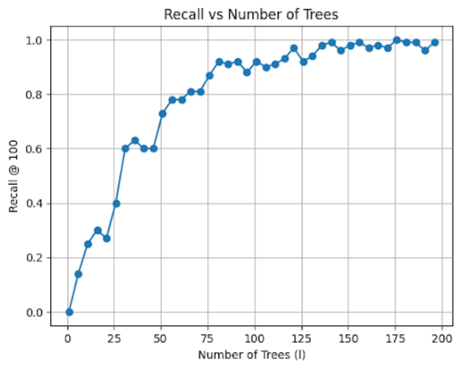


Figure 2: Recall for a Single LSH Forest with 10,000 Vectors.

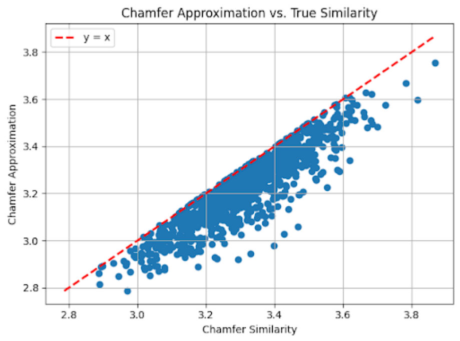


Figure 3: Approximations for 1,000 documents a random query.

5.2 Performance on Real-World Data

In contrast, the performance on real-world data (specifically, the HotpotQA dataset) showed a significantly different trend. Despite increasing the number of trees in the LSH forest, Recall@100 remained low and failed to show a clear upward trend. Even with the maximum number of trees tested (l = 100), the Recall@100 score plateaued far below 1.0. This result can be attributed to several key factors inherent to the complexity of real-world data:

5.2.1 High Dimensional Noise

The vector representations used for documents and queries were based on pre-trained transformer embeddings (768-dimensional). High-dimensional spaces are known to suffer from the ”curse of dimensionality,” where the distinction between near and far vectors becomes less meaningful. This effect was likely exacerbated by the presence of noisy or redundant dimensions in the embeddings. We note that dimensionality reduction techniques which preserve pair wise distances may help overcome this problem.

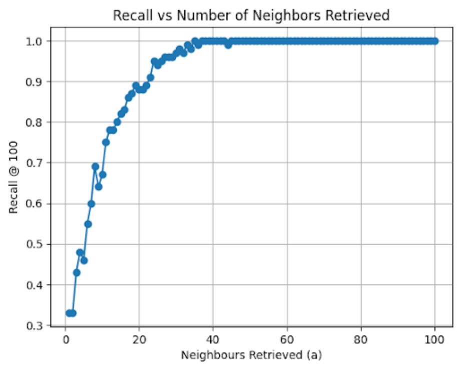


Figure 4: Recall for Multi-vector Retrieval on Simulated Data.

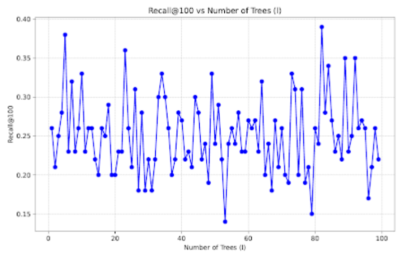


Figure 5: Recall for the HotPotQA dataset.

5.2.2 Semantic Mismatch and Query Complexity

The queries in the HotpotQA dataset are inherently complex, often requiring multi-hop reasoning across multiple sentences or paragraphs. The LSH forest, which relies on vector similarity, is not well-suited for capturing such complex semantic relationships.

5.2.3 LSH Forest Parameters

The LSH forest itself may not have been optimally configured for the real-world data. The parameters used were found to be performing well on our simulated dataset, but factors such as the depth of the forest (km), the split ratio, and the hash attempt limit may need to be re-optimized for this dataset. The high-dimensional nature of the data may require a different configuration than the one used for the simulated setup, which was something we decided was beyond the scope of this project.

6 Depth-Weighted Chamfer Approximation Algorithm

Sections 3-5 demonstrated that the LSH Forest is effective for approximating Chamfer similarity, primarily by improving the quadratic scale of the chamfer similarity nearest neighbor search. However, in practice, even highly expressive multi-vector embedding models represent documents using a manageable number of vectors, while the total number of documents tends to be extremely large. Thus, creating a separate LSH forest for each document is impractical. This observation motivated the development of an algorithm that approximates Chamfer similarity efficiently by significantly reducing dependence both on the vector set size per document and the total number of documents.

In this section, we present an algorithm that leverages the probabilistic similarity inherent in node collisions within LSH forests. The core idea is to use a depth-weighted voting scheme that approximates Chamfer similarity by taking advantage of the exponential decay in collision probability as a function of node depth. Specifically, vectors that collide deeper in the tree are likely to be more similar, a behavior described by the following probabilistic model:

6.1 Theoretical Motivation

Consider two normalized vectors $\mathbf{x}, \mathbf{y} \in \mathbb{S}^{d-1}$. For random hyperplane LSH (SimHash), the probability that these vectors fall on the same side of a random hyperplane is given by:

$$\Pr[h(\mathbf{x}) = h(\mathbf{y})] = 1 - \frac{\arccos(\langle \mathbf{x}, \mathbf{y} \rangle)}{\pi} \tag{10}$$

Concatenating multiple hash functions, the probability of collision at depth ℓ behaves approximately as:

$$\Pr[\text{collision at depth } \ell] \approx e^{-\lambda r^2 \ell}, \tag{11}$$

where $r = |\mathbf{x} - \mathbf{y}|$. This exponential decay indicates that collisions at deeper levels strongly indicate greater similarity.The approximation in Eq.(11) follows by linearising the arccos term for small angular separations, i.e., $\arccos(\langle \mathbf{x}, \mathbf{y} \rangle) \approx \frac{\pi}{2} - \langle \mathbf{x}, \mathbf{y} \rangle$, and noting that for unit vectors $r^2 = |\mathbf{x} - \mathbf{y}|^2 = 2\big(1 - \langle \mathbf{x}, \mathbf{y} \rangle\big)$. Concatenating k independent hash bits and grouping constants then yields the Gaussian–like decay $e^{-\lambda r^2 \ell}$ used above.

6.2 Algorithm

Our depth-weighted scoring algorithm operates as follows:

1. For each node in the LSH forest, maintain a count dictionary tracking the number of vectors reaching that node. The root node contains all vectors with an initial count of 1, while leaf nodes typically have sparse dictionaries with non-zero counts only for vectors reaching them.

2. Define a depth-dependent weight function $w(\ell)$, which assigns greater weight to collisions occurring deeper in the tree. *Here, L denotes the maximum depth of any tree in the forest, and k is a tunable midpoint (typically $k \approx L/2$) that sets the inflection of the logistic curve. We explored three families of weight functions:

   - **Logistic:** $w_{\log}(\ell) = \dfrac{1}{1+e^{\alpha(\ell-k)}}$, where $\alpha > 0$ and typically $k \approx L/2$.
   - **Linear:** $w_{\text{lin}}(\ell) = \dfrac{L-\ell+1}{L+1}$.
   - **Exponential:** $w_{\exp}(\ell) = \gamma^{\ell}$ with $0 < \gamma < 1$.

   Empirical simulations indicate that, although the logistic form was usually the most stable choice, all three weighting families yielded high retrieval quality once their hyper-parameters are tuned to the data. In practice, the parameters (α, k), γ, and the depth limit L can be selected automatically by minimizing a validation loss (e.g., cross-entropy on held-out queries) or even learned end-to-end as part of the retrieval model's objective function.

   At query time, for each query vector q, traverse the path from the root to its corresponding leaf node, and recursively aggregate scores when ascending back to the root:

   - At the leaf node (depth D), initialize the score with the count dictionary at depth D, weighted by $w(D)$.
   - Ascend recursively toward the root, at each step calculating a unique count by subtracting counts already encountered from the node's total counts. Aggregate these new contributions using the depth-dependent weighting $w(\ell)$.
   - Stop either upon reaching the root or when incremental scores fall below a predefined threshold.
   - Average scores obtained from multiple trees to obtain the final approximation of the Chamfer similarity.

6.3 Runtime Analysis

Let $|Q|$ be the number of query vectors, T the number of trees in the LSH forest, and L the maximum depth of any tree. Because each query vector is processed once in every tree, the total running time is

$$\mathcal{O}\big(|Q|\, T\, L\big).$$

When the forest is (approximately) balanced, the depth satisfies

$$L \;=\; \Theta\big(\log(N\,\bar{P})\big),$$

Algorithm 1 Depth-Weighted Chamfer Approximation

```
1: procedure ApproxChamfer($Q$, forest, $w(\ell)$)
2:     Initialize finalScores dictionary
3:     for all $q \in Q$ do
4:         Initialize scoreDict, seenCounts
5:         Descend to leaf node for $q$, at depth $D$
6:         scoreDict $\leftarrow w(D)\cdot$ counts at leaf
7:         seenCounts $\leftarrow$ leaf counts
8:         for $\ell = D - 1$ to 0 do
9:             Compute uniqueCounts = counts[$\ell$] - seenCounts
10:            scoreDict += $w(\ell)\cdot$ uniqueCounts
11:            seenCounts += uniqueCounts
12:            Break loop if incremental contribution < threshold
13:        end for
14:        Add scoreDict to finalScores
15:    end for
16:    Average finalScores over all query vectors and trees
17:    return final similarity scores
18: end procedure
```

where N is the number of documents and $\bar{P}$ is the average number of vectors per document (so $N\bar{P}$ is the total number of indexed vectors). Hence the overall query time becomes

$$\mathcal{O}\big(|Q|\, T\, \log(N\,\bar{P})\big),$$

a logarithmic improvement over the naïve $\mathcal{O}\big(|Q|\, N\, \bar{P}\big)$ Chamfer computation.

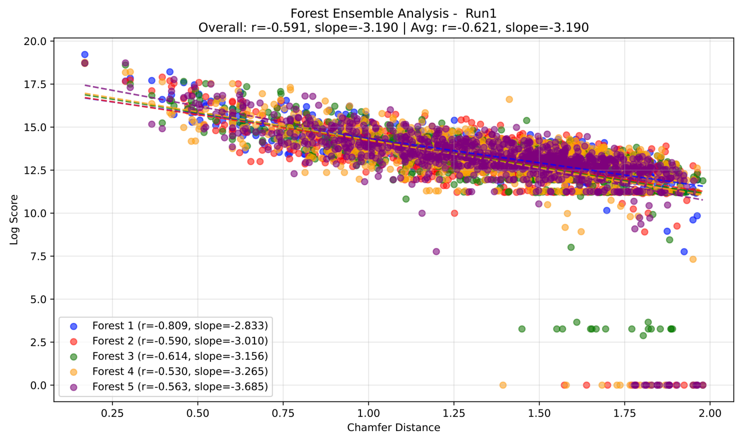
Updates to the per-document score dictionary are performed in $\mathcal{O}(1)$ time per accessed document ID, because the set of keys is fixed after indexing and can be stored in a pre-allocated array or constant-time hash map.

6.4 Empirical Results and Discussion

Simulations using randomly generated data confirm that this depth-weighted scoring algorithm significantly correlates with the true Chamfer similarity scores. Figure 6 illustrates the correlation coefficients and the regression slope, indicating substantial predictive capability. Notably, the true Chamfer nearest neighbors consistently receive the highest scores, distinguishing themselves from other documents in our tests.

In the simulations reported in Figure 6, each document and query was generated as a single random Gaussian vector, and its multi-vector representation was obtained by adding independent Gaussian noise to that prototype. Run 1 used a noise variance of 0.10, whereas in Run 2 we reduced the variance to 0.05. A thorough evaluation on real-world collections remains future work. The two independent runs had 5 forests, each with 5 trees. Each forest is built upon

1000 independent documents, each with 50 vectors, and is queried by a set of 50 random vectors with the same parameters.



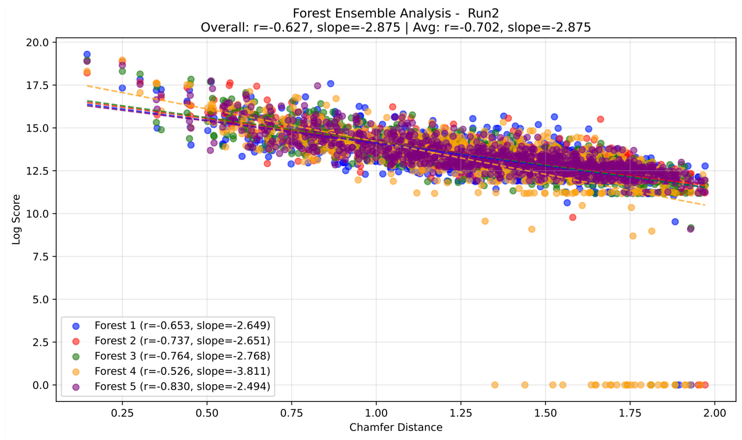


Figure 6: Forest behavior across two different setting of within doc variance.

Conclusion

This work introduced a new perspective on multi-vector information retrieval by treating Chamfer (MaxSim) search as an approximate nearest- neighbour problem in an *LSH-Forest*. We made two main contributions:

1. **Chamfer approximation using LSH fFrest.** We proved and tested that an LSH-Forest equipped with $L = \Theta\big(\log(N\bar{P})\big)$ depth and T trees yields a $(1-\varepsilon)$ multiplicative approximation of the true Chamfer similarity in $\mathcal{O}\big(|Q|\,T\,\log(N\bar{P})\big)$ time.

2. **Depth-weighted scoring algorithm.** We proposed a collision-aware voting scheme that leverages node depth to sidestep the explicit max-pool required by Chamfer.

In short, our study shows that depth-aware LSH-Forests provide a viable approximation to Chamfer similarity with improved runtime, yet a systematic exploration of high-dimensional embeddings, automatic parameter tuning, and large-scale benchmarks is needed to fully evaluate the use cases of the methods we explored.

References

[1] Alexandr Andoni and Ilya Razenshteyn. *Optimal Data-Dependent Hashing for Approximate Near Neighbors*. 2015. arXiv: `1501.01062 [cs.DS]`. URL: `https://arxiv.org/abs/1501.01062`.

[2] Alexandr Andoni et al. "Practical and optimal LSH for angular distance". In: *NIPS*. 2017.

[3] Ainesh Bakshi et al. *A Near-Linear Time Algorithm for the Chamfer Distance*. 2023. arXiv: `2307.03043 [cs.DS]`. URL: `https://arxiv.org/abs/2307.03043`.

[4] Mayank Bawa, Tyson Condie, and Priya Ganesan. "LSH Forest: Self-Tuning Indexes for Similarity Search". In: *WWW*. 2005.

[5] Laxman Dhulipala et al. *MUVERA: Multi-Vector Retrieval via Fixed Dimensional Encodings*. 2024. arXiv: `2405.19504 [cs.DS]`. URL: `https://arxiv.org/abs/2405.19504`.

[6] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 39–48. ISBN: 9781450380164. DOI: `10.1145/3397271.3401075`. URL: `https://doi.org/10.1145/3397271.3401075`.

[7] Keshav Santhanam et al. *ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction*. 2022. arXiv: `2112.01488 [cs.IR]`. URL: `https://arxiv.org/abs/2112.01488`.

[8] Keshav Santhanam et al. "PLAID: An Efficient Engine for Late Interaction Retrieval". In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. CIKM '22. Atlanta, GA, USA: Association for Computing Machinery, 2022, pp. 1747–1756. ISBN: 9781450392365. DOI: `10.1145/3511808.3557325`. URL: `https://doi.org/10.1145/3511808.3557325`.