НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
сентябрь–октябрь 2025 Том 25 № 5 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
September–October 2025 Vol. 25 No 5 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2025-25-5-902-909

Vector embeddings compression using clustering with the ensemble of oblivious decision trees and separate centroids storage

Nikita A. Tomilov✉

ITMO University, Saint Petersburg, 197101, Russian Federation

programmer174@icloud.com✉, https://orcid.org/0000-0001-9325-0356

Abstract

The modern approach to search textual and multimodal data in large collections involves the transformation of the documents into vector embeddings. To store these embeddings efficiently different approaches could be used, such as quantization, which results in loss of precision and reduction of search accuracy. Previously, a method was proposed that reduces the loss of precision during quantization. In that method, clustering of the embeddings with *k*-Means algorithm is performed, then a bias, or delta, being the difference between the cluster centroid and vector embedding, is computed, and then only this delta is quantized. In this article a modification of that method is proposed, with a different clustering algorithm, the ensemble of Oblivious Decision Trees. The essence of the method lies in training an ensemble of binary Oblivious Decision Trees. This ensemble is used to compute a hash for each of the original vectors, and the vectors with the same hash are considered as belonging to the same cluster. In case when the resulting cluster count is too big or too small for the dataset, a reclustering process is also performed. Each cluster is then stored using two different files: the first file contains the per-vector biases, or deltas, and the second file contains identifiers and the positions of the data in the first file. The data in the first file is quantized and then compressed with a general-purpose compression algorithm. The usage of Oblivious Decision Trees allows us to reduce the size of the storage compared to the same storage organization with *k*-Means clustering. The proposed clustering method was tested on Fashion-MNIST-784-Euclidean and NYT-256-angular dataset against the *k*-Means clustering. The proposed method demonstrates a better compression quality compared to clustering via *k*-Means, demonstrating up to 4.7 % less storage size for NF4 quantization for Brotli compression algorithm. For other compression algorithms the storage size reduction is less noticeable. However, the proposed clustering algorithm provides a bigger error value compared to *k*-Means, up to 16 % in the worst-case scenario. Compared to Parquet, the proposed clustering method demonstrates a lesser error value for the Fashion-MNIST-784-Euclidean dataset when using quantizations FP8 and NF4. For the NYT-256-angular dataset, compared to Parquet, the proposed method allows better compression for all tested quantization types. These results suggest that the proposed clustering method can be utilized not only for the nearest neighbor search applications, but also for compression tasks, when the increase in the quantization error can be ignored.

Keywords

vector representations, embeddings, oblivious decision tree, clustering, compression of vector embeddings

For citation: Tomilov N.A. Vector embeddings compression using clustering with the ensemble of oblivious decision trees and separate centroids storage. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2025, vol. 25, no. 5, pp. 902–909. doi: 10.17586/2226-1494-2025-25-5-902-909

УДК 004.021

Сжатие векторных представлений с использованием кластеризации с помощью ансамбля небрежных решающих деревьев и раздельного хранения центроидов

Никита Андреевич Томилов✉

Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация

programmer174@icloud.com✉, https://orcid.org/0000-0001-9325-0356

Аннотация

Введение. Современные подходы к поиску текстовых и мультимодальных данных в больших коллекциях предполагают преобразование документов в векторные представления. Для эффективного хранения этих векторов применяется квантизация, которая снижает точность представления и, как следствие, ухудшает качество поиска. Ранее был предложен метод, уменьшающий потери точности при квантизации, в котором векторы кластеризуются с помощью алгоритма *k*-средних, вычисляется смещение, или дельта, являющееся разницей между центроидом кластера и исходным вектором, после чего квантуется только смещение. В настоящей работе предлагается модификация этого метода, использующая другой алгоритм кластеризации — ансамбль небрежных решающих деревьев. **Метод.** Разработанный метод основывается на обучении ансамбля бинарных небрежных решающих деревьев. Этот ансамбль используется для вычисления хэша каждого исходного векторного представления, после чего векторные представления с одинаковым хэшем рассматриваются как принадлежащие одному кластеру. В случае, если число результирующих кластеров слишком большое или слишком маленькое для используемого набора данных, производится процесс перекластеризации. Каждый кластер сохраняется в двух отдельных файлах: первый содержит смещения для каждого вектора, второй — идентификаторы и позиции данных в первом файле. Данные в первом файле подвергаются квантизации и затем сжимаются с помощью универсального алгоритма сжатия. **Основные результаты.** Предложенный метод кластеризации протестирован на наборах данных fashion-mnist-784-euclidean и NYT-256-angular и сравнивался с кластеризацией методом *k*-средних. Метод показал лучшее качество сжатия, демонстрируя до 4,7 % меньшее занимаемое дисковое пространство при использовании квантизации NF4 и алгоритма сжатия Brotli. Для других алгоритмов сжатия увеличение пространства оказалось менее значительным. Однако представленный алгоритм кластеризации демонстрирует большее значение показателя ошибки квантизации по сравнению с методом *k*-средних, минимум до 16 %. По сравнению с форматом Parquet разработанный метод кластеризации продемонстрировал меньший показатель ошибки для набора данных fashion-mnist-784-euclidean при использовании квантизаций FP8 и NF4. Для набора данных NYT-256-angular по сравнению с Parquet предложенный метод при использовании алгоритма сжатия Brotli позволяет добиться лучшего качества сжатия для всех протестированных типов квантизации. **Обсуждение.** Полученные результаты свидетельствуют о том, что разработанный метод кластеризации может быть использован не только в задачах поиска ближайших соседей, но и в задачах сжатия данных, когда увеличением ошибки квантизации можно пренебречь.

Ключевые слова

векторные представления, эмбеддинг, небрежное решающее дерево, кластеризация, сжатие векторных представлений

Ссылка для цитирования: Томилов Н.А. Сжатие векторных представлений с использованием кластеризации с помощью ансамбля небрежных решающих деревьев и раздельного хранения центроидов // Научно-технический вестник информационных технологий, механики и оптики. 2025. Т. 25, № 5. С. 902–909 (на англ. яз.). doi: 10.17586/2226-1494-2025-25-5-902-909

Introduction

One of the effective strategies for organizing search in large collections of textual and multimodal data involves transforming documents into vector embeddings, being sequences of floating-point numbers that encode semantic content [1]. In this framework, machine learning techniques are used to construct such embeddings for both the documents and the search queries. Efficient search is then performed via nearest-neighbor retrieval in the resulting high-dimensional space, using precomputed indices [2]. Storing uncompressed embeddings in high-precision formats such as FP32 requires significant memory and disk space. To address this, quantization methods are often employed to convert values from FP32 to lower-precision formats such as FP16 or FP8 [3], enabling substantial space savings. However, this comes at the cost of reduced accuracy, which may negatively affect downstream applications, including further machine learning tasks [4].

In [5] authors explored vector embeddings compression with the help of *k*-Means clustering, where we cluster embeddings and separately store the centroids and per-vector biases (deltas). Each embedding can thus be represented as a pair of the index of its closest centroid and the delta vector between the embedding and the centroid. These components can be compressed with different levels of precision, for instance, storing centroids in FP32 to preserve accuracy while applying quantization to the deltas using FP16 or FP8. In [6] authors explored the clustering algorithm via the ensemble of Oblivious Trees, which proved to be beneficial for improving the approximate nearest neighbor search.

In scope of this research, we combine these previous works and present the usage of the clustering algorithm using the ensemble of Oblivious Trees for compressing various kinds of vector embeddings via the method of storing centroids and deltas separately [5]. Our hypothesis suggests that usage of this clustering algorithm will improve the compression compared to *k*-Means [7] clustering.

Methods

Description of the clustering method used

In our previous study [6] a clustering algorithm was proposed using an ensemble of binary Oblivious Decision Trees (ODTs) [8]. To manage computational costs, each tree is trained on a random subsample, having size N, taken from the original dataset of size N_{dataset}, such that $N = N_{\text{dataset}} \times TrainRatio$, where $TrainRatio$ is between 0.1 and 1. We transform each original vector embedding into multiple low-dimensional variants by randomly selecting subsets of its components. This results in M so-called subvectors, having dimension d_m, per each of the original vector embedding. These subvectors are essentially different views of the data, each used to train a separate tree. A mapping table is maintained to associate the indices of the projected components with their positions in the original vector, allowing reconstruction during ensemble creation.

Each ODT partitions the input space hierarchically. At each tree level j, we select a component K_j and a threshold X_j splitting the data into two groups depending on whether the value of the selected component is less than or equal to X_j, or bigger than X_j. The rule by which the component and threshold is selected is called a partitioning rule and is a hyper-parameter of the algorithm, and in scope of our research we find the values that perform the partition to minimize the variance of distances between each vector embedding and the average vector embedding for that group.

The same splitting rule is applied across all branches at a given level, which is a defining property of oblivious trees. This process repeats until the tree reaches the desired depth *depth*. Due to the nature of the splits, it is possible for some groups to be empty at deeper levels, which is expected and does not affect the structure.

After training, each tree encodes a vector as a binary string of zeros and ones, depending on the comparison outcome on each tree level with respect to the corresponding threshold X_j. This binary string is effectively a hash for each vector. This hash is called *SH* and has length of *depth*. With M trees, each vector receives M such hashes which are then concatenated into a single composite hash string *TH*. This final hash effectively clusters vectors, as those with the same composite hash are placed into the same group, or cluster.

It is worth mentioning that this method does not have precise control over the resulting number of clusters and can lead to a huge number of them, in the worst-case scenario being comparable to the amount of vector embeddings in the dataset. Since the maximum resulting cluster number is $2^{M\,depth}$, having 5 trees of depth 4 can result in 1,048,576 clusters. This is inappropriate for datasets having comparable or lower numbers of vectors. To mitigate this problem, we introduce so-called reclustering with the parameter F and threshold T. If the maximum resulting number of clusters R_1 is less than T, we perform the clustering as normal, and then split each of the resulting clusters up to F sub-clusters with the different clustering algorithm, such as *k*-Means, so that the maximum number of resulting sub-clusters is $R_1 \times F$. We call this process reclustering up. If the maximum resulting number of clusters R_2 is more than T, we pick the centroids of the resulting clusters, and then cluster those centroids using different clustering algorithms, such as *k*-Means, with such parameters, that the maximum number of resulting sub-clusters is R_2/F. We then use these new centroids to cluster the original dataset via *k*-Nearest Neighbors algorithm. We call this process reclustering down.

The example resulting clusters count for $T = 4{,}000$ are presented in Table 1.

As a result of this reclustering we gain more precise control over the maximum number of clusters. It is worth mentioning that such reclustering is only necessary for relatively small datasets and is not necessary for such combinations of M and *depth* so that the resulting maximum number of clusters is significantly less than the number of vectors in the source dataset.

This research has four important differences compared to [6]. Firstly, in this research we focus on data compression instead of approximate nearest neighbor search. Secondly, we were able to optimize the training phase of the ODTs, so that now we can train the ODTs on more embeddings, specifically 15 % of the original dataset ($TrainRatio = 0.15$), instead of the first five thousand vectors of the dataset as it was in the previous research. Also, we focus only on binary trees, since ternary trees proved to create too many clusters for compression purposes, and introduce reclustering to mitigate the issue with having too many clusters.

Description of the storing method used

To validate the compression quality of the Oblivious Tree clustering algorithm compared to *k*-Means clustering algorithm, we use the storage method [5] with some key differences. The essence of the method is to store

Table 1. Reclustering for $T = 4{,}000$

Number of trees M	Depth of each tree (*depth*)	Maximum number of clusters without reclustering	Maximum number of clusters after reclustering up with $F = 10$	Maximum number of clusters after reclustering down with $F = 30$
4	2	256	2,560	—
4	3	4,096	—	137
4	4	65,536	—	2,185
5	2	1,024	10,240	—
5	3	32,768	—	1,092
5	4	1,048,576	—	34,953

clustered embeddings in binary files, so that each cluster is represented by two files. The first binary file, called page file, contains entries for a given document index, where an entry contains the metadata, such as segment index, followed by the serialized floating-point values array. This array is generated by subtracting the original vector embedding value from the centroid of the respective cluster. During serialization, this floating-point values array can be quantized from their original FP32 representation via any scalar quantization, such as FP8 [9], to reduce space.

The second binary file, called the page index file, serves as an index, mapping each primary identifier to its byte offset in the page file, indicating where the entry for a given document index is. All offsets in this index file, as well as indexes in the page file, are encoded using variable length encoding to reduce overhead. After the page is finalized, meaning there are no more vector embeddings for this page, the page file is compressed using any general-purpose compression algorithm, to further reduce space.

To convert the dataset to the proposed storage, it is necessary to cluster the vector embeddings and generate the page files and page index files as described above. To retrieve the vector embedding for a given document index, it is necessary to traverse page index files to find the precise page that contains the necessary embedding, then decompress the necessary page file, and fetch the entry for the retrieved offset. After that, it is necessary to deserialize the floating-point array, containing the vector embedding delta, and sum it with the respective vector centroid to restore the original vector embedding.

It is worth mentioning that this storage organization method can be used with different clustering algorithms, which is why we can use it to compare Oblivious Tree clustering against *k*-Means clustering in the context of vector embeddings compression while having the storage layer similar for both clustering algorithms.

The key difference between the method presented in this research and what was explored in [5] is that in this research the whole page file is compressed, while in [5] the compression was used for each vector embedding individually. Working with the full page will improve compression as there will be more data to find repetitive patterns to utilize during compression. However, it also increases the vector retrieval time, which is out of the scope of the research. On top of that, NF4 [10] is also considered as another quantization method that can pack a floating-point value to just 4 bits.

Storage implementation

To perform the experiments, we implemented the proposed storage solution in Kotlin programming language. To serialize the entries, we used an Apache Avro library. Each of the entries contains a byte array that was constructed by converting the delta value directly from the FP32 representation to specified quantization format to one of the target quantizations, namely FP16, FP8 and NF4, reducing the storage space from four bytes per the component of delta representation to two, one or half of a byte respectively. The centroids of the clusters are stored separately, serialized via the built-in Java serialization mechanism, in their original FP32 representation. As an additional metadata, the entry contains a segment index that is always zero for the datasets used to conduct the experiment.

To convert the datasets to store them using the proposed storage implementation, we perform clustering of the dataset via the *k*-Means algorithm, and via the ensemble of the Oblivious Trees. Then, vectors belonging to each of the clusters get converted into the page files. After that, the page files are compressed using the following compression algorithms: LZMA, Zstandard, Brotli [11]. To use these algorithms in Kotlin, the Apache Commons Compress[1], zstd-jni[2] and brotli4j[3] libraries were used. For Zstandard and Brotli, the compression level value was set to 22 and 11, respectively. For LZMA, the default parameters provided by the library were used.

Results

Experiment setup

To verify the compression quality when using the Oblivious Tree clustering algorithm, we selected the same two datasets as before, NYT-256-angular[4] and Fashion-MNIST-784-Euclidean [12], taken from the popular ANN-Benchmarks [13] collection. These datasets, as well as others from the same collection, are often used to benchmark average nearest neighbor vector search algorithms. They differ from each other in the way the data was originally created.

These datasets were clustered using the following clustering algorithms:

1. *k*-Means clustering, with the maximum number of clusters K being an arbitrary number between 1,000 and 15,000.
2. ODT ensemble with the following parameters:
   a. $M = 5$;
   b. $d_m = 196$ for the Fashion-MNIST dataset, 32 for the NYT dataset (meaning, 25 % from the source dataset dimensionality for Fashion-MNIST and 12.5 % for NYT);
   c. $depth = 2, 4$;
   d. $TrainRatio = 0.15$ %;
   e. $F = 10$ for $depth = 2$ and 30 for $depth = 4$;
   f. $T = 4{,}000$.

For both clustering algorithms we then calculate the delta values and quantize them to their FP16, FP8 and NF4 representations, after which the data is compressed using LZMA, ZStd and Brotli algorithms. The centroids are stored in their original FP32 representation and are not compressed. The exact FP8 quantization type used is E3M4, meaning 3 bits for the exponent, 4 bits for the mantissa, and one sign bit.

[1] Commons Compress — Overview. URL: https://commons.apache.org/proper/commons-compress/index.html (accessed: 11.05.2025).

[2] luben/zstd-jni: JNI binding for Zstd. URL: https://github.com/luben/zstd-jni (accessed: 11.05.2025).

[3] hyperxpro/Brotli4j: Brotli4j provides Brotli compression and decompression for Java. URL: https://github.com/hyperxpro/Brotli4j (accessed: 11.05.2025).

[4] Newman D. Bag of Words // UCI Machine Learning Repository. 2008. URL: https://archive.ics.uci.edu/dataset/164/bag+of+words (accessed: 11.05.2025). DOI: 10.24432/C5ZG6P

Upon creating the storage files, we measure the storage size represented as the sum of the sizes of each of the individual files generated for each of these clustering methods, against the error value, represented as Euclidean or Angular distance, depending on the dataset used, between the retrieved vector embedding and the original vector embedding. Assuming the vector embedding of size n, retrieved from the storage and having the component values (vc_0, …, vc_n), as vc, and the original vector embedding of the same size n, having component values (vs_0, …, vs_n), as vs, the error value e, being the distance metric, can be calculated using the following equations:

— Euclidean distance

$$e = \sqrt{\sum_{k=0}^{n} (vc_k - vs_k)^2},$$

— Angular distance

$$e = \frac{\sum_{k=0}^{n} vc_k vs_k}{\sqrt{\sum_{k=0}^{n} vc_k^2} \sqrt{\sum_{k=0}^{n} vs_k^2}}.$$

The lower the metric, the closer the retrieved vector embedding to the original one, meaning the lesser the error caused by lossy compression via quantization.

We then select only the best two parameter combinations for the k-Means and Oblivious Tree clustering, judging by the lowest possible storage size, and provide the values for all combinations of compression algorithm and quantization type for these value combinations. To provide a baseline, we also store the clustered values in Parquet [14] format in both lossless (float array) and lossy (byte arrays of quantized float arrays) for the specified compression algorithms supported by Parquet with compression levels configured similarly to the proposed solution.

The test setup has the following specifications: AMD Ryzen 7 7700X 8C16T; 64 GB RAM; NVMe WD SN850X 2TB; OS Ubuntu 22.04; OpenJDK 22; a tool for comparing vector search algorithms, developed in previous research [15], using Java Microbenchmark Harness [16].

Experiment results

The results for the Fashion-MNIST-784-Euclidean dataset are presented in Table 2. The vector embeddings in this dataset are grayscale 28 by 28-pixel images of clothing having the values from 0 to 255, with a lot of values being zero indicating the black pixels, making this a sparse dataset. The size of the original dataset, excluding the data that was not part of the experiment like precomputed closest neighbors, in HDF5 file without any compression and in the original FP32 format is 179.44 MB. This value is also present in the table as the Original HDF5 file row. The notation for the ODT clustering corresponds to $M/d_m/depth/F/\{up, down\}$, where M, d_m, $depth$, F are the parameters specified above, and *up* or *down* indicate reclustering direction.

Table 2. Storage size and error value (as Euclidean distance) for Fashion-MNIST-784-Euclidean dataset, for different compression algorithms and quantizations

Storage	Quantization	Storage size for compression algorithm, MB				Error value
		Without compression	LZMA	ZStd	Brotli	
Original HDF5 file	FP32	179.44	—	—	—	—
Parquet	FP32	34.81	—	26.78	27.42	—
	FP16	91.43	—	32.88	34.11	0.02 ± 0.01
	FP8	46.56	—	18.17	18.65	85.23 ± 29.23
	NF4	24.12	—	9.53	10.02	380.86 ± 211.83
k-Means, K = 1,000	FP32	182.48	29.18	32.59	30.83	—
	FP16	93.58	44.42	47.17	44.24	0.25 ± 0.20
	FP8	48.73	20.44	21.29	20.39	55.04 ± 13.29
	NF4	26.25	12.21	12.63	12.24	155.90 ± 50.78
k-Means, K = 10,000	FP32	182.49	29.44	33.14	30.80	—
	FP16	94.79	45.54	48.10	45.17	0.24 ± 0.18
	FP8	49.93	21.66	22.39	21.57	53.36 ± 14.26
	NF4	27.50	13.53	13.78	13.47	152.80 ± 47.02
ODT, 5/192/4/30/down	FP32	182.55	29.18	32.22	30.91	—
	FP16	92.88	43.48	47.20	43.36	0.29 ± 0.13
	FP8	48.02	19.90	21.12	19.80	60.91 ± 8.99
	NF4	25.58	11.63	12.14	11.66	169.57 ± 32.90
ODT, 5/192/2/10/up	FP32	183.48	31.10	34.84	31.90	—
	FP16	93.85	45.00	49.25	44.47	0.24 ± 0.13
	FP8	48.98	20.77	21.98	20.69	54.41 ± 12.80
	NF4	26.54	12.74	13.11	12.57	151.71 ± 37.53

For this dataset, the Oblivious Tree clustering method demonstrates worse compression compared to *k*-Means for original FP32 vector embedding representations, but better compression when using delta storage with quantization. This can be seen for both selected parameter combinations for both LZMA and Brotli compression algorithms, and for the first parameter combination for ZStd compression algorithm. For example, for FP16 quantization and Brotli compression, the Oblivious Tree clustering requires 43.36 MB of size in the best-case scenario, compared to 44.24 MB for *k*-Means clustering. It means that Oblivious Tree clustering can use 0.88 MB less compared to *k*-Means clustering. Relative to the storage size of *k*-Means clustering this difference equals to around 2 % less storage size. For NF4 quantization and Brotli compression, the values are 12.24 and 11.66 MB respectively, which is 0.58 MB less storage size, or 4.7 % less relative to *k*-Means clustering.

This advantage is offset by the fact that Oblivious Tree clustering demonstrates bigger error value, showing bigger loss during quantization. For the combinations mentioned above, the average error value for ODTs is 0.04 and 14 more than for *k*-Means clustering, or 16 % and 9 % more relative to *k*-Means clustering, respectively. On top of that, for this dataset, when using original FP32 vector representations and without delta storage both clustering methods cannot demonstrate better compression compared to Parquet, which is able to effectively encode repeating zero values in the vector embeddings. This encoding is so effective that its lossless compression is better than the lossy compression for FP16 encoding for the proposed method and both clustering algorithms. Overall, using ODT clustering combined with delta storage cannot offer better compression compared to quantized Parquet storage, however it can demonstrate less error value for some of the quantizations. Compared to Parquet, using ODTs can lower the error value from around 85 and 380 for FP8 and NF4 quantizations, respectively, to around 61 and 170 for the parameter combinations demonstrating the least storage size.

The results for the NYT-256-angular dataset are presented in Table 3. The vector embeddings in this dataset are text embeddings of the NYT articles, having dimensionality 256 and all the values between –0.342 and 0.346, making it a dense dataset. As the angular distance is usually smaller, in the table the error value is multiplied by 1,000. The size of the original dataset, excluding the data that was not part of the experiment like precomputed closest neighbors, in HDF5 file without any compression and in the original FP32 format is 283.21 MB.

For this dataset, the results are comparable to the results of the previous dataset, represented in Table 2. The Oblivious Tree clustering method demonstrates better compression when using delta storage with quantization. For example, for FP8 quantization and Brotli compression,

Table 3. Storage size and error value (as angular distance) for NYT-256-angular dataset, for different compression algorithms and quantizations

Storage	Quantization	Storage size for compression algorithm, MB				Error value, multiplied by 1.000
		Without compression	LZMA	ZStd	Brotli	
Original HDF5 file	FP32	283.21	—	—	—	—
Parquet	FP32	289.82	—	244.85	239.05	—
	FP16	156.95	—	133.18	133.53	0.00 ± 0.00
	FP8	85.14	—	47.89	46.12	6.94 ± 0.89
	NF4	49.73	—	28.58	27.98	74.70 ± 12.58
k-Means, K = 1,000	FP32	298.27	239.04	239.20	237.14	—
	FP16	157.65	124.43	126.95	121.89	0.00 ± 0.00
	FP8	86.80	45.71	45.01	44.88	6.93 ± 1.49
	NF4	51.37	28.27	27.06	26.92	70.12 ± 14.35
k-Means, K = 10,000	FP32	298.44	240.24	239.73	238.29	—
	FP16	166.48	134.46	136.47	131.74	0.00 ± 0.00
	FP8	95.74	55.30	53.96	53.64	7.27 ± 4.24
	NF4	60.28	39.14	36.24	35.93	63.90 ± 34.65
ODT, 5/32/4/30/down	FP32	298.62	239.76	239.67	237.90	—
	FP16	157.18	124.55	125.03	121.82	0.00 ± 0.00
	FP8	86.32	45.19	46.94	44.00	7.44 ± 1.31
	NF4	50.87	28.68	28.32	26.18	73.27 ± 8.88
ODT, 5/32/2/10/up	FP32	303.23	244.36	244.34	242.29	—
	FP16	161.83	128.82	131.26	126.21	0.00 ± 0.00
	FP8	90.97	50.60	49.80	49.62	6.53 ± 1.13
	NF4	55.48	33.13	31.68	31.47	71.18 ± 15.58

the Oblivious Tree clustering requires 44 MB of size in the best-case scenario, compared to 44.88 MB for *k*-Means clustering, which is 0.88 MB less storage size or almost 2 % less relative to *k*-Means clustering. For NF4 quantization and Brotli compression, the values are 26.18 and 26.92 MB respectively, which is 0.74 MB less or is around 2.7 % less size relative to *k*-Means clustering. For FP16 compression the difference between compression algorithms is negligible. However, for this dataset both clustering methods demonstrate better compression than Parquet for the same compression algorithm.

The error value for the FP16 encoding for this dataset is almost zero, making FP16 a good choice for this dataset, allowing the storage size to significantly decrease without compromising the precision. As for more lossy quantizations, the results are also the same as for the previous dataset. The Oblivious Tree quantization also has a bigger error value. For the combinations mentioned above, the average error value for ODTs is 0.00051 and 0.00315 more than for *k*-Means clustering, or 7.4 % and 4.5 % more relative to *k*-Means clustering, respectively. Compared to Parquet, for this dataset ODT clustering demonstrates better compression with comparable error value. For NF4 quantization, ODT clustering achieves an error value of up to about 73, while for Parquet the value is around 75. For this quantization and Brotli compression, Parquet requires 27.98 MB of storage, which is 1.8 MB more than for ODT clustering, or 6.8 % more relative to ODT clustering. Overall, using ODT clustering in combination with delta storage can offer better compression compared to quantized Parquet storage with a comparable error value.

Conclusion

In this work, we proposed a storage solution for vector embeddings that combines oblivious decision tree clustering with delta encoding and quantization. Our experiments demonstrated that this method reduces storage requirements compared to *k*-Means clustering, with savings of up to 4.7 % for FP8 and NF4 quantization. We also showed its loss in precision compared to *k*-Means, which indicates that the method is most effective when storage efficiency is prioritized over absolute precision.

The main advantages of the method are the improved compression through delta quantization, the retention of centroids in FP32 precision, and the possibility of fine-tuning cluster counts via reclustering. Its limitations, namely greater retrieval time and significant training cost, are inherited from the underlying clustering algorithm and storage implementation. It is also worth mentioning that the reclustering step does not allow determining the target cluster solely based on the hash value of the embedding, negating the sharding benefits mentioned in [6]. It is possible to achieve sharding when reclustering up is being performed, however, in this case ODT can help determine only the target node, rather than target cluster of vector embeddings on the node, which would be possible without reclustering. We consider these trade-offs acceptable for applications where compact storage is critical such as large-scale embedding databases.

The experiment results highlight that while oblivious tree clustering is beneficial for compression tasks where an increase in quantization error is acceptable, for sparse datasets like Fashion-MNIST-784-Euclidean, Parquet format with lossless compression can outperform both ODT clustering and *k*-Means clustering methods when using original FP32 representations, due to its effectiveness in encoding repeating zero values, and shows better compression for lossy quantizations. For dense datasets like NYT-256-angular, both clustering methods demonstrate better compression than Parquet for the same compression algorithm and same quantization. Additionally, for the NYT-256-angular dataset, FP16 encoding yields an almost zero error value, making it a suitable choice for significant space reduction without compromising precision.

The introduced reclustering step allows for better control of the resulting number of clusters, improving upon one of the drawbacks of ODT clustering, at the cost of limiting the sharding possibilities. Other advantages and disadvantages of both the delta storage and ODT clustering remain unchanged.

These results suggest that the proposed clustering method can be utilized not only for the nearest neighbor search applications, as already explored in [6], but also for storage and compression of the vector embeddings for later use, such as machine learning tasks over the previously gathered embeddings, when the quantization error can be ignored. Further advancements in this area include experimenting with other partitioning rules to better group the vector embeddings, achieving better compression.

References

1. Grbovic M., Cheng H. Real-time personalization using embeddings for search ranking at airbnb. *Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 311–320. https://doi.org/10.1145/3219819.3219885
2. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z. Fast nearest neighbor search in medical image databases. *Proc. of the 22th International Conference on Very Large Data Bases*, 1996, pp. 215–226. https://doi.org/10.5555/645922.673493
3. Zhou W., Lu Y., Li H., Tian Q. Scalar quantization for large scale image search. *Proc. of the 20th ACM International Conference on Multimedia*, 2012, pp. 169–178. https://doi.org/10.1145/2393347.2393377
4. Zhang J., Yang J., Yuen H. Training with low-precision embedding tables. *Systems for Machine Learning Workshop at NeurIPS*, 2018, vol. 2018.

Литература

1. Grbovic M., Cheng H. Real-time personalization using embeddings for search ranking at airbnb // Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018. P. 311–320. https://doi.org/10.1145/3219819.3219885
2. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z. Fast nearest neighbor search in medical image databases // Proc. of the 22th International Conference on Very Large Data Bases. 1996. P. 215–226. https://doi.org/10.5555/645922.673493
3. Zhou W., Lu Y., Li H., Tian Q. Scalar quantization for large scale image search // Proc. of the 20th ACM International Conference on Multimedia. 2012. P. 169–178. https://doi.org/10.1145/2393347.2393377
4. Zhang J., Yang J., Yuen H. Training with low-precision embedding tables // Systems for Machine Learning Workshop at NeurIPS. 2018. V. 2018.

5. *Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V.* A method of storing vector data in compressed form using clustering. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2024, vol. 24, no. 1, pp. 112–117. (in Russian). https://doi.org/10.17586/2226-1494-2024-24-1-112-117
6. Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V. Vector search using method of clustering using ensemble of oblivious trees. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2025, vol. 25, no. 2. pp. 339–344. https://doi.org/10.17586/2226-1494-2025-25-2-339-344
7. Kanungo T., Mount D., Netanyahu N., Piatko Ch., Silverman R., Wu A. The analysis of a simple k-means clustering algorithm. *Proc. of the 16th Annual Symposium on Computational Geometry*, 2000, pp. 100–109. https://doi.org/10.1145/336154.336189
8. Lou Y., Obukhov M. BDT: gradient boosted decision tables for high accuracy and scoring efficiency. *Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1893–1901. https://doi.org/10.1145/3097983.3098175
9. Kuzmin A., van Baalen M., Ren Y., Nagel M., Peters J., Blankevoort T. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 2022, vol. 35, pp. 14651–14662.
10. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QloRA: efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*, 2023, vol. 36, pp. 1–5.
11. Alakuijala J., Farruggia A., Ferragina P., Kliuchnikov E., Obryk R., Szabadka Z., Vandevenne L. Brotli: A general-purpose data compressor. *ACM Transactions on Information Systems (TOIS)*, 2018, vol. 37, no. 1, pp. 1–30. https://doi.org/10.1145/3231935
12. Xiao H., Rasul K., Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, 2017, arXiv:1708.07747. https://doi.org/10.48550/arXiv.1708.07747
13. Aumüller M., Bernhardsson E., Faithfull A. ANN-Benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. *Lecture Notes in Computer Science*, 2017, vol. 10609, pp. 34–49. https://doi.org/10.1007/978-3-319-68474-1_3
14. Zeng X., Hui Y., Shen J., Pavlo A., McKinney W., Zhang H. An empirical evaluation of columnar storage formats. *Proc. of the VLDB Endowment*, 2023, vol. 17, no. 2, pp. 148–161. https://doi.org/10.14778/3626292.3626298
15. Turov V.P., Tomilov N.A., Babayants A.A. Developing a tool to compare vector search algorithms. *Proc. of the 11th Congress of Young Scientists*, 2022, vol. 1, pp. 446–450. (in Russian)
16. Laaber C., Leitner P. An evaluation of open-source software microbenchmark suites for continuous performance assessment. *Proc. of the 15th International Conference on Mining Software Repositories (MSR '18)*, 2018, pp. 119–130. https://doi.org/10.1145/3196398.3196407

5. Томилов Н.А., Туров В.П., Бабаянц А.А., Платонов А.В. Метод хранения векторных представлений в сжатом виде с применением кластеризации // Научно-технический вестник информационных технологий, механики и оптики. 2024. Т. 24. № 1. С. 112–117. https://doi.org/10.17586/2226-1494-2024-24-1-112-117
6. Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V. Vector search using method of clustering using ensemble of oblivious trees // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2025. V. 25. N 2. P. 339–344. https://doi.org/10.17586/2226-1494-2025-25-2-339-344
7. Kanungo T., Mount D., Netanyahu N., Piatko Ch., Silverman R., Wu A. The analysis of a simple k-means clustering algorithm // Proc. of the 16th Annual Symposium on Computational Geometry. 2000. P. 100–109. https://doi.org/10.1145/336154.336189
8. Lou Y., Obukhov M. BDT: gradient boosted decision tables for high accuracy and scoring efficiency // Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017. P. 1893–1901. https://doi.org/10.1145/3097983.3098175
9. Kuzmin A., van Baalen M., Ren Y., Nagel M., Peters J., Blankevoort T. Fp8 quantization: The power of the exponent // Advances in Neural Information Processing Systems. 2022. V. 35. P. 1–10.
10. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QloRA: efficient finetuning of quantized LLMs // Advances in Neural Information Processing Systems. 2023. V. 36. P. 1–5.
11. Alakuijala J., Farruggia A., Ferragina P., Kliuchnikov E., Obryk R., Szabadka Z., Vandevenne L. Brotli: A general-purpose data compressor // ACM Transactions on Information Systems (TOIS). 2018. V. 37. N 1. P. 1–30. https://doi.org/10.1145/3231935
12. Xiao H., Rasul K., Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms // arXiv. 2017. arXiv:1708.07747. https://doi.org/10.48550/arXiv.1708.07747
13. Aumüller M., Bernhardsson E., Faithfull A. ANN-Benchmarks: a benchmarking tool for approximate nearest neighbor algorithms // Lecture Notes in Computer Science. 2017. V. 10609. P. 34–49. https://doi.org/10.1007/978-3-319-68474-1_3
14. Zeng X., Hui Y., Shen J., Pavlo A., McKinney W., Zhang H. An empirical evaluation of columnar storage formats // Proc. of the VLDB Endowment. 2023. V. 17. N 2. P. 148–161. https://doi.org/10.14778/3626292.3626298
15. Туров В.П., Томилов Н.А., Бабаянц А.А. Разработка инструмента сравнения алгоритмов векторного поиска // XI Конгресс молодых учёных: сборник научных трудов. СПб: федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО». 2022. Т. 1. С. 446–450.
16. Laaber C., Leitner P. An evaluation of open-source software microbenchmark suites for continuous performance assessment // Proc. of the 15th International Conference on Mining Software Repositories (MSR '18). 2018. P. 119–130. https://doi.org/10.1145/3196398.3196407

Author

Nikita A. Tomilov — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, sc 57225127284, https://orcid.org/0000-0001-9325-0356, programmer174@icloud.com

Автор

Томилов Никита Андреевич — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, sc 57225127284, https://orcid.org/0000-0001-9325-0356, programmer174@icloud.com

Received 11.05.2025
Approved after reviewing 29.08.2025
Accepted 21.09.2025

Статья поступила в редакцию 11.05.2025
Одобрена после рецензирования 29.08.2025
Принята к печати 21.09.2025