Individualized non-uniform quantization for vector search

Mariano Tepper
mariano.tepper@ibm.com
IBM

Ted Willke
ted.willke@ibm.com
IBM

Abstract

Embedding vectors are widely used for representing unstructured data and searching through it for semantically similar items. However, the large size of these vectors, due to their high-dimensionality, creates problems for modern vector search techniques: retrieving large vectors from memory/storage is expensive and their footprint is costly. In this work, we present NVQ (non-uniform vector quantization), a new vector compression technique that is computationally and spatially efficient in the high-fidelity regime. The core in NVQ is to use novel parsimonious and computationally efficient nonlinearities for building non-uniform vector quantizers. Critically, these quantizers are *individually* learned for each indexed vector. Our experimental results show that NVQ exhibits improved accuracy compared to the state of the art with a minimal computational cost.

1 Introduction

Modern embedding models are very adept at creating high-dimensional vectors whose spatial similarities reflect the semantic affinities between their inputs (e.g., images, audio, video, text, genomics, and computer code [15, 27, 34, 42, 43]). Many applications [e.g., 11, 12, 21, 29, 35] have used this capability to find semantically relevant results in massive collections of vectors by retrieving the approximate nearest neighbors (ANN) of a given query vector. The most notable for its real-world deployment is retriever-augmented generation (RAG) [32], using ANN to ground knowledge and to prevent LLMs from hallucinating.

Despite the outstanding progress in similarity search in the last few years [e.g., 4, 18, 22, 36, 46], modern indices still struggle with high-dimensional vectors [2, 47]. Although moderate dimensionalities ($D \approx 128$) are handled accurately and with high performance [2], efficiency suffers when dealing with the dimensionalities typically produced by embedding models ($D \approx 768$ and beyond) [47]. Next, we present two noteworthy examples of this behavior.

For in-memory indices (where the vectors and the index itself are held in main memory), graph-based search [e.g., 5, 18, 36, 46] is currently the state-of-the-art. Here, a directed graph, where each vertex corresponds to a database vector and edges represent neighbor-relationships between vectors, is efficiently traversed to find the approximate nearest neighbors in sub-linear time. As we traverse the graph, vectors are fetched from memory in a random-like access pattern [2], causing a memory bandwidth bottleneck. This high memory utilization drastically increases the memory latency [45] to access each vector, ultimately leading to suboptimal search performance. Even masterful software engineering [2] cannot hide the increased latency.

DiskANN [46] and SPANN [14] are good representatives of the state of the art in the space of on-disk indices (where the vectors and the index itself are held in SSDs). DiskANN is a graph-based method that stores the high-dimensional vectors on disk next to the adjacency list of each node. With this layout, the same 4KB disk block can be used to store the list and the vector. However, this design does not hold for vectors with 1024 dimensions or more since the vector itself occupies at least 4KB: here the vector payload does not come for free. SPANN is an inverted index that stores each vector in ~8 clusters, with each cluster arranged contiguously on disk. Whereas this approach works for moderate-dimensional vectors, where each cluster can have a moderate size of 48KB [14], it creates an explosion of disk footprint and bandwidth utilization for high-dimensional vectors.

In addition, just storing large collections of high-dimensional vectors is a problem in itself; e.g., a relatively small collection of 10 million 1536-dimensional vectors occupies 57GB in single-precision format. Existing

vector compression techniques fall short as they are incompatible with random access patterns or do not provide sufficient accuracy [2, 4, 22]. New vector representations are needed that simultaneously present the following characteristics: (1) reduced footprint to alleviate memory/storage pressure and costs, (2) similarities must be preserved, and (3) similarity computations must be fast.

We tackle these problems by introducing **NVQ** (pronounced "new vec"). NVQ is a non-uniform vector quantization method that is fitted to each vector *individually* (in contrast to most vector quantizers that are fitted to the entire collection of vectors) to create more faithful representations (i.e., in terms of reconstruction error and similarities between vectors). Non-uniformity is embodied in NVQ through the novel use of parsimonious and computationally efficient nonlinearities, whose parameters are learned for each vector separately. This work presents the following detailed contributions:

- NVQ is the first method to learn a nonlinear quantizer individually for each vector. We also present an optimization algorithm for this learning problem. We show that this formulation improves on the commonly used uniform quantization by a significant margin in terms of reconstruction error and vector search accuracy. Furthermore, we show that additional gains are obtained by dividing each vector into subvectors and individually applying NVQ to them. The code is available at https://github.com/marianotepper/nuveq.
- We propose the use of the Kumaraswamy distribution and of the logistic/logit nonlinearities for vector quantization. We also introduce a lightweight approximation of the logistic/logit nonlinearities that relies on not-quite transcendental functions. These nonlinearities are characterized by only two scalars and thus conduct to highly parsimonious and computationally efficient quantizers.
- We use NVQ in a production environment as part of an SSD-index for vector search. Here, NVQ reduces storage use by more than 3x versus 32-bit full-precision vectors with an impact of less than 0.01 on recall above 0.95. The code is available as part of JVector at https://github.com/datastax/jvector/.

The remainder of the work is structured as follows. We first review the related work in Section 2. In Section 3 we present NVQ, the optimization problem behind it and the algorithm to solve it. In Section 4, we present three nonlinear functions that power NVQ, discussing the technical merits of each one in terms of expressiveness and efficiency. We then present extensive experimental results in Section 5 and provide concluding remarks in Section 6.

2 Related work

Research on approximate nearest neighbor (ANN) search has grown rapidly, motivated by the demands of applications like retrieval augmented generation, recommendation systems, and hybrid search, which involve increasingly larger datasets, higher dimensionality, and stringent recall and latency requirements. ANN methods in the literature fall into different categories. We focus the analysis on techniques suitable for large, high-dimensional datasets. Tree-based approaches, such as [10, 13, 38, 44], struggle with the curse of dimensionality and do not generally scale well. Hashing-based approaches, such as [24, 25, 50], scale well but tend to suffer lower recall or use data-dependent hashing requiring full re-indexing or complex management pipelines as vectors are added. Graph-based approaches, such as [17, 18, 36, 41, 46], typically provide better trade-offs between latency and recall than hashing and tree-based methods. Quantization-based methods, such as [2, 3, 4, 7, 19, 20, 26, 30, 40, 52, 57], the category most relevant to our work, have become central due to their ability to compress large vector sets while supporting fast and accurate retrieval. Recent studies also propose hybrid algorithms, combining strengths from multiple categories to improve overall performance [2, 22, 26, 46]. In this work, we present a new approach to quantization and evaluate it as a reranker in conjunction with a graph-based technique. Comprehensive reviews and tutorials on ANN techniques can be found in recent surveys [6, 16, 33, 39, 51, 53].

Quantization reduces vector size by reducing the precision of each dimension, which saves memory and storage footprint while lowering bandwidth utilization. Scalar is widely used due to its computational simplicity. The most common form of scalar quantization is uniform quantization, in which a range of vector values are divided into equal intervals that are each assigned a single value. This simplicity makes it fast but also introduces large reconstruction errors, which lowers recall. Locally-adaptive quantization is a recent advancement in vector search that allows for the quantization parameters fitted individually at the vector level. The first example of this for vector search is Locally-adaptive Vector Quantization [2], where the

data is first globally centered and then uniform scalar quantization is applied based on the minimum and maximum values within each vector. NVQ is able to further reduce reconstruction error by dividing vectors into subvectors and fitting each more precisely through the use of simple nonlinear functions.

Vector quantization that takes into account the data distribution can lower reconstruction error for the same level of compression. PQ [26] is the most common distribution-aware technique. The quantizer applies k-means clustering to M multi-dimensional subspaces of the input vector, encoding each subvector as the ID of the nearest centroid (e.g., an 8-bit unsigned integer). PQ encoding can be trained using a small dataset sample. It offers fast codebook generation, high compression ratios, and modest reconstruction error. However, the resulting recall is generally too low to use it without reranking. Additionally, as mentioned in [2], PQ was designed primarily for inverted indices in which a precomputed table of partial similarities can be stored in transposed form. This enables efficient SIMD computation of distances between a query and multiple database vectors simultaneously. This transposition is not compatible with the random access patterns seen in graph-based search. NVQ does not offer the compression ratios that PQ does, but it offers lower reconstruction error at levels of compression that are acceptable for storage and is easily vectorized for SIMD. The Anisotropic Vector Quantization introduced in [22] and used in ScaNN[1] is also a learned quantizer that builds on PQ, inheriting the same benefits and limitations.

Aggressively compressing vectors to save memory often comes with a substantial reduction in recall. A full-precision copy of the vectors may be retained in storage and used to boost recall in two-level quantization or with a final reranking step, as in [46, 48]. PQ codebooks and residuals are used in [56] to boost recall, but the joint optimization of these levels is computationally prohibitive at large scale and the recall is lower than methods that utilize full-precision vectors. SPANN in [14] uses an inverted index, keeping only centroids in memory and storing multiple copies of vectors (associated with multiple clusters) on disk. When used for storage-based reranking, NVQ significantly reduces the storage footprint and the amount of data transferred during I/O operations, benefitting these methods and others that store high-fidelity vectors on disk.

3 Non-uniform vector quantization

A quantizer takes continuous values in a domain $\mathcal{D}$ and discretizes them by mapping them to values in the set $\{0, 1, 2, \ldots, 2^\beta - 1\}$ of natural numbers, where β is the number of bits of the representation. Formally, the β-bit quantizer $Q : \mathcal{D} \to \mathcal{B}_\beta$ and the β-bit dequantizer $Q^{-1} : \mathcal{B}_\beta \to \mathcal{D}$ are defined as

$$Q(x; h, \theta, \beta) := \left\lfloor (2^\beta - 1) h(x; \theta) + 1/2 \right\rfloor \tag{1}$$

$$Q^{-1}(y; h, \theta, \beta) := h^{-1}\left((2^\beta - 1)^{-1} y; \theta \right), \tag{2}$$

where $h : \mathcal{D} \to [0, 1]$ is an invertible nonlinearity with parameters $\theta \in \mathcal{C}$. Although Q is not an invertible function due to the floor function, we use the notation Q^{-1} to denote its "lossy inverse" (its inverse when $\beta \to \infty$).

Of critical importance in this setting is the parsimony of the parameters θ. In one extreme, we can achieve lossless quantization if we store the histogram of the values in the vector. Of course, this would not be sensible since transmitting $\{Q(x_i; h, \theta, \beta)\}$ would be more expensive than transmitting $\mathbf{x}$ directly. On the other extreme, a uniform quantizer uses h as the identity function, i.e., $h(x) = x$, and $\theta = \varnothing$. The main question of this work is: can we improve upon the uniform quantization using a data-driven approach with a highly parsimonious parameter set? We answer this question in the affirmative, showing that consistent improvements can be achieved using nonlinearities with only two scalar parameters. In fact, this is not only possible, but computationally very efficient.

Traditionally, non-uniform quantizers are constructed by using the empirical CDF (i.e., its cumulative histogram) of the data. Here, we would use h as an invertible parametric curve fitted to the empirical CDF. With this approach, we would allocate more bits to the most common data values. However, this is a problem for encoding vectors in the context of vector search, where we are less interested in the fidelity of the vectors themselves. What truly matters is fidelity of the similarity between the vectors and the queries.

Without loss of generality, we illustrate this point with maximum inner product (MIP) search, where the similarity between a query vector $\mathbf{q}$ and a database vector $\mathbf{x}$ is given by their dot product $\langle \mathbf{q}, \mathbf{x} \rangle$ and we seek

[1] https://github.com/google-research/google-research/tree/master/scann

to recover the vectors in $\mathcal{X}$ that are most similar to the query. Extending these concepts to other popular similarity measures, such as Euclidean distance or cosine similarity is straightforward. In MIP,

$$\langle \mathbf{q}, \mathbf{x} \rangle \approx \langle \mathbf{q}, \widetilde{\mathbf{x}} \rangle, \quad \text{where} \quad \begin{aligned} \mathbf{y} &= Q(\mathbf{x}; h, \theta, \beta), \\ \widetilde{\mathbf{x}} &= Q^{-1}(\mathbf{y}; h, \theta, \beta). \end{aligned} \tag{3}$$

Here, the encoder/decoder are applied entry-wise for some appropriate selection of h, θ, β, producing a quantized version $\widetilde{\mathbf{x}}$ of $\mathbf{x}$. Clearly, the entries of $\mathbf{x}$ with larger magnitudes have a larger impact on the dot product than those close to zero. However, we observe that modern embedding models produce vectors whose values present bell-shaped empirical value distributions, centered approximately at zero. Allocating more bits to these common values would yield a small pay-off.

Assuming a flat prior on the angular distribution of the queries (the norm of the query has no effect in MIP), we have [22] for a d-dimensional vector $\mathbf{x} = [x_1, \cdots, x_d]^\top$,

$$\mathrm{E}_{\mathbf{q}} \left[(\langle \mathbf{q}, \mathbf{x} \rangle - \langle \mathbf{q}, \widetilde{\mathbf{x}} \rangle)^2 \right] = \|\mathbf{x} - \widetilde{\mathbf{x}}\|_2^2 = \sum_{i=1}^{d} (x_i - \widetilde{x}_i)^2 . \tag{4}$$

Thus, minimizing the reconstruction error is the best approach for learning quantizers without additional information about the queries.

Given a suitable nonlinearity h, we seek the best parameters θ for each $\mathbf{x} \in \mathcal{X}$. For this, we solve

$$\min_{\theta \in \mathcal{C}} \ell_h(\theta) \quad \text{where} \quad \ell_h(\theta) := \sum_{i=1}^{d} \left(x_i - Q^{-1} \left(Q(x_i; h, \theta, \beta); h, \theta, \beta \right) \right)^2 . \tag{5}$$

Notice that we perform no dataset-level optimization. Each vector is treated individually, finding the parameters that minimize its own reconstruction error.

We found that more informative loss values and stopping conditions can be achieved by working with the improvement over the uniform quantization baseline instead of the raw reconstruction error. That is, we solve

$$\max_{\theta \in \mathcal{C}} \frac{\ell_{\text{UNIF}}(\varnothing)}{\ell_h(\theta)}, \tag{6}$$

where $\ell_{\text{UNIF}}(\varnothing)$ denotes the reconstruction error obtained with the parameterless uniform quantization. With this formulation, a value higher or lower than 1 indicates an improvement or decline in the quality of our quantization, respectively. It is important to note that Problem (6) is $|\theta|$-dimensional. Its runtime depends on d for a d-dimensional vector $\mathbf{x}$ but not its parameter space.

Solving Problem (6) individually for each vector in the database has multiple benefits. First, it can be carried seamlessly when inserting each vector in the index. Second, it is completely robust to distribution shifts in the data that may occur over time [1, 8]. Finally, and maybe more importantly, it enables "overfitting" the quantizer to each vector without downsides.

3.1 The optimization

The nonlinearities h and h^{-1} make Problem (6) non-convex. Moreover, the objective function ℓ_h is discontinuous due to the floor function. This technical difficulty is commonly bypassed by using a straight-through estimator [9, 49]. In this work, we follow a gradient-free optimization approach, as described next.

With the nonlinearities introduced in Section 4, Problem (6) is two-dimensional and we observe in Figure 1 that its landscape has a relatively large basin. Thus, it is amenable to gradient-free optimization techniques.[2] In this work, we optimize Problem (6) using separable natural evolution strategies [55], which estimate a proxy of the gradient by computing the objective function at stochastic samples around the current estimate of the solution. In Algorithm 1 we modify Algorithm 6 in [55] to account for the constraint $\theta \in \mathcal{C}$ in Problem (6). The modification consists of adding a projection step in Line 5. This step takes different forms depending on the nonlinearity, as described in Section 4.

[2] Alternatively, the two-dimensional nature of the problem allows to find a solution using grid search.

Algorithm 1: Separable NES with constraints

Inputs: $f : \mathbb{R}_+^m \to \mathbb{R}$, $\boldsymbol{\mu}_{\text{init}} \in \mathbb{R}_+^m$, $\sigma_{\text{init}} \in \mathbb{R}_+^m$, $\eta_{\boldsymbol{\mu}} \in \mathbb{R}_+$, $\eta_\sigma \in \mathbb{R}_+$, $T \in \mathbb{N}_1$.
Output: The solution $\boldsymbol{\mu}$.

$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}_{\text{init}}$
repeat
  for $k = 1, \ldots, T$ **do**
    draw sample $\mathbf{s}_k \sim \mathcal{N}(0, \mathbf{I})$
    $\mathbf{z}_k \leftarrow \text{project}_{\mathcal{C}}\left(\boldsymbol{\mu} + \sigma \odot \mathbf{s}_k\right)$ // $\odot$ represents the Hadamard product
  compute utilities u_k for each $\mathbf{z}_k$ with respect to $f(\mathbf{z}_k)$ [55, section 3.1]
  compute natural gradients $\begin{cases} \nabla_{\boldsymbol{\mu}} J \leftarrow \sum_{k=1}^{T} u_k \cdot \mathbf{s}_k \\ \nabla_\sigma J \leftarrow \sum_{k=1}^{T} u_k \cdot (\mathbf{s}_k^2 - 1) \end{cases}$
  update parameters $\begin{cases} \boldsymbol{\mu} \leftarrow \max\left\{\boldsymbol{\mu} + \eta_{\boldsymbol{\mu}} \sigma \cdot \nabla_{\boldsymbol{\mu}} J, \mathbf{0}\right\} \\ \sigma \leftarrow \sigma \exp(\eta_\sigma / 2 \cdot \nabla_\sigma J) \end{cases}$
until *stopping condition is met*

In Algorithm 1, we use $f(\cdot) = \ell_{\text{UNIF}}(\varnothing)/\ell_h(\cdot)$. For the hyperparameters, we approximately adopt the suggestions in [55]: the number of stochastic samples is $T = 2(4 + \lfloor 3 \log |\theta| \rfloor)$ and the learning rates are $\eta_{\boldsymbol{\mu}} = 1$ and $\eta_\sigma = (9 + 3 \log |\theta|)/(5m\sqrt{|\theta|})$. The initial values $\boldsymbol{\mu}_{\text{init}}$ and σ_{init} are specific to each nonlinearity, as described in Section 4. Unless specified, we use the stopping condition $\left|\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}^{(t-1)}\right| < 10^{-4}$, where $\boldsymbol{\mu}^{(t)}$ is the value of $\boldsymbol{\mu}$ at the t-th iteration. We also impose a minimum of 10 iterations.

3.2 Quantizing subvectors

Inspired by PQ [26], we consider dividing a vector into subvectors. We divide its d dimensions at random into m sets, forming m subvectors with d/m dimensions each, i.e., $\mathbf{x} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}]$. In this case, we optimize Problem (6) for each subvector individually and we have $\theta = \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(m)}\}$. With modern embedding models, using $m \in \{2, 4, 8\}$ ensures that d is a multiple of m.

The payload of quantizing subvectors is obviously higher than that of quantizing entire vectors. However, this cost is relatively small when compared to the size of the vector themselves. For 1024-dimensional vectors, it the vector values occupy 1024 bytes for $\beta = 8$. As we describe in the next section, for $m \in \{2, 4, 8\}$, we have $|\theta| = \{8, 16, 32\}$ parameters ($|\theta^{(j)}| = 4$ for $j = 1, \cdots, m$), which occupy 32B, 64B, and 128B, respectively. We finally point out that it is often desirable to store the vectors in having a cache-aligned memory layout [2]. From this point of view, the parameters can be stored in the "free space" used to produce this alignment.

4 The nonlinearities

We now introduce the nonlinear functions used in Problem (6). These functions, which are used for vector quantization for the first time in this work, are characterized by their parsimony, requiring only two scalar parameters, and their computational efficiency. We present three alternatives that strike different trade-offs between expressiveness and computational efficiency, as shown in Table 1.

Each compressed vector needs two additional values in its payload, $x_{\min} = \min_i x_i$ and $x_{\max} = \max_i x_i$. These two values are not optimized as part of Problem (6), they are used in different ways to normalize the each set of nonlinearities as described in each subsection.

4.1 The generalist: The Kumaraswamy distribution

The wide use of the beta distribution is a testament of its expressiveness to model different phenomena in the interval $[0, 1]$. However, it is computationally expensive as its partition function if given by the beta function, that lacks a closed form. The Kumaraswamy distribution is similar to the beta distribution in expressiveness, see Figure 2, but much more computationally amenable since its probability density, cumulative distribution, and quantile functions can be expressed in closed form [28, 31, 54]. Its cumulative

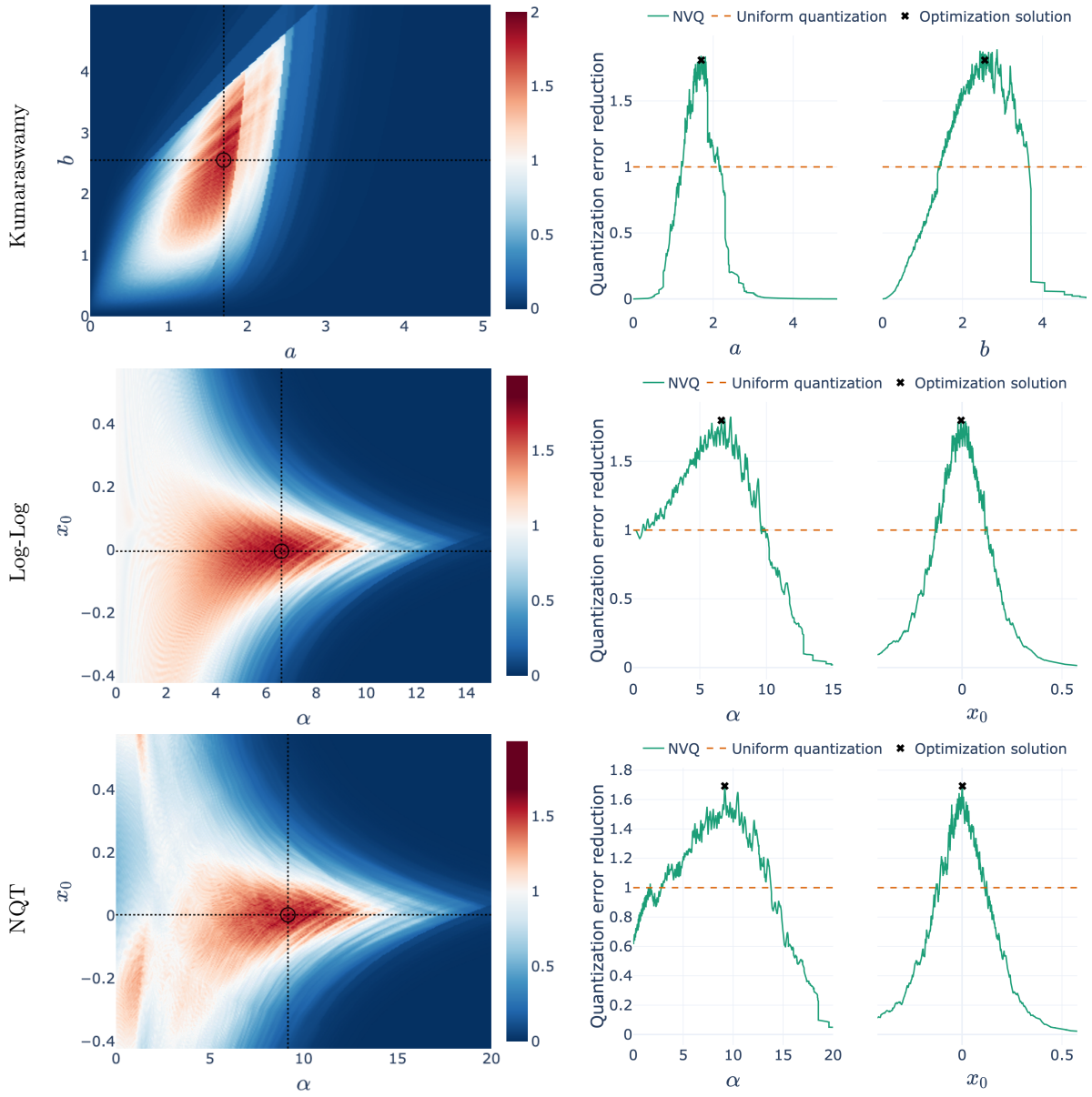

Figure 1: Algorithm 1 finds a good maximum of Problem (6) using the nonlinearities presented in Section 4 for $\beta = 8$ bits (similar results for $\beta = 4$ and a different dataset are provided in Figures 12 to 14 of the appendix). **Left:** the landscape of the objective function in Problem (6), which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), as a function of the nonlinearity parameters for one vector from ada002-100k (see Table 2). The solution found by Algorithm 1 is marked by a black circle. **Right:** two cross cuts taken at the values corresponding to the found solution.

Table 1: Expressiveness and computational cost of the three nonlinearities used in this work. We measure the cost in terms of fundamental functions and main computer instructions (for simplicity, we do not count additions, subtractions, or bit operations). While the Kumaraswamy distribution is the most expressive nonlinearity (see Figure 2), its "heavy" use of fundamental functions makes it computationally demanding. The other two nonlinearities are less expressive (but sufficiently so for vectors from embedding models) and computationally nimbler. Implementation details in Appendices A and B.

		Kumaraswamy	Logistic/Logit	NQT Logistic/Logit
Expressiveness		multiple distributions	bell-shaped	bell-shaped
Encoding cost	exp	2	1	0
	log	2	0	0
	FMA	18	6	2
	MUL	8	2	0
	DIV	0	1	1
Decoding cost	exp	2	0	0
	log	2	1	0
	FMA	18	7	2
	MUL	8	2	0
	DIV	0	1	1

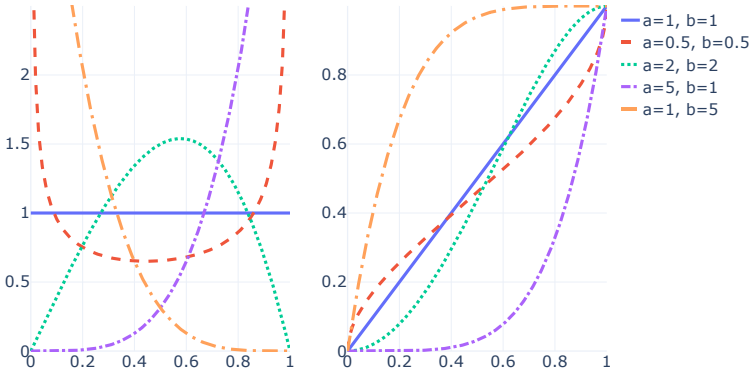

Figure 2: The PFD (left) and CDF (right) of the Kumaraswamy distribution, defined in Equation (7), for different values of a and b. The Kumaraswamy distribution allows to represent sample distributions with a variety of shapes.

distribution and its quantile functions are respectively given by

$$F_{\text{KS}}(x; a, b) := 1 - (1 - x^a)^b, \tag{7}$$

$$F_{\text{KS}}^{-1}(y; a, b) := \left(1 - (1 - y)^{1/b}\right)^{1/a}, \tag{8}$$

where $x \in [0, 1]$ and $a, b \in \mathbb{R}_+$. The Kumaraswamy distribution is traditionally defined in the open interval $(0, 1)$. Since F is well defined for $x = 0$ and $x = 1$, we extend the range to the closed interval. To operate in $[0, 1]$, we normalize the values x_i in $\mathbf{x}$ as $(x_i - x_{\min})/(x_{\max} - x_{\min})$, where $x_{\min} = \min_i x_i$ and $x_{\max} = \max_i x_i$.

We propose to solve Problem (6) with the parametrization

$$\theta := [a, b], \qquad h := F_{\text{KS}}, \qquad h^{-1} := F_{\text{KS}}^{-1}. \tag{9}$$

Using this parametrization in Equations (1) and (2) makes the pair Q, Q^{-1} highly adaptable with only two parameters (the scalars a, b) and involves a closed-form computation.

For the Kumaraswamy nonlinearity, $a, b \in \mathbb{R}_+$. The projection in Line 5 of Algorithm 1 takes the form

$$\text{project}_{\mathcal{C}}(a) := \max\{a, 0\}, \qquad \text{project}_{\mathcal{C}}(b), := \max\{b, 0\}, \tag{10}$$

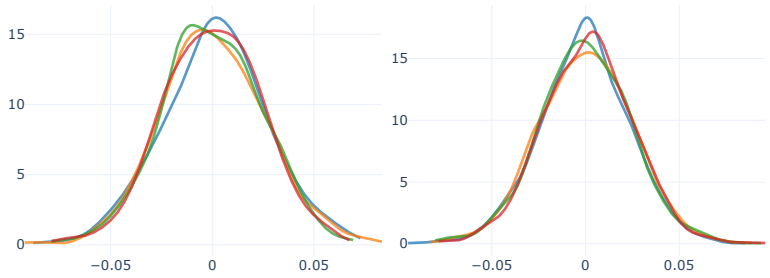

Figure 3: The kernel density estimate of PDF of the values from four different vectors in gecko-100k (left) and in openai-v3-100k (right). The sample PDFs are bell-shaped and, although they are all generally similar, they are not exactly the same. We can improve the quantization by tuning it for each vector individually.

and we use the initial values $\boldsymbol{\mu}_{\text{init}} = [1, 1]$ and $\sigma_{\text{init}} = [1, 1]$.

Efficiency: We implement the power operator using the identity $x^c = \exp(c \log x)$. We note that numerically stable formulations have been recently proposed [54] but we did not encounter such problems in our gradient-free optimization algorithm. The logarithm and exponential functions can be implemented with sufficient accuracy using minimax polynomial approximations. Additional implementation details are provided in Appendix A. These highly flexible nonlinearities are relatively fast to compute, as observed in Table 1.

4.2 The specialist: The scaled logistic and scaled logit

The Kumaraswamy distribution imbues NVQ with a high degree of flexibility. However, the computation of powers is an expensive operation that does not commonly have native support in CPUs or GPUs. Using exponentials and logarithms, as described in Appendix A, alleviates but does not eliminate this problem.

By examining the empirical distribution of the values in vectors produced by modern embedding models, we see in Figure 3 that they are bell shaped. Thus, we are not fully exploiting the Kumaraswamy distribution's flexibility: an optimal solution to Problem (6) for these vectors would never include parameters that generate exponential-like distributions even if the Kumaraswamy distribution supports this choice.[3] Therefore, seeking computational efficiency, we turn our attention to nonlinearities that are specifically well suited to handle bell-shaped distributions.

The standard logistic and logit functions on $\mathbb{R} \to (0, 1)$ and $(0, 1) \to \mathbb{R}$, respectively, are defined as

$$\text{logistic}(x; \alpha, x_0) := (1 + \exp(-\alpha(x - x_0)))^{-1}, \tag{11}$$

$$\text{logit}(y; \alpha, x_0) := \alpha^{-1} \log\left(\frac{y}{1-y}\right) + x_0. \tag{12}$$

Let $x_{\min} = \min_i x_i$ and $x_{\max} = \max_i x_i$. We are interested in quantizing values in $[x_{\min}, x_{\max}]$. For this, we define scaled versions of these functions, i.e., $\text{logistic}_{\text{SCALED}} : [x_{\min}, x_{\max}] \to [0, 1]$ and $\text{logit}_{\text{SCALED}} : [0, 1] \to [x_{\min}, x_{\max}]$, as follows

$$\text{logistic}_{\text{SCALED}}(x; \alpha, x_0) := \frac{\text{logistic}(\delta^{-1}x; \alpha, x_0) - \text{logistic}(\delta^{-1}x_{\min}; \alpha, x_0)}{\Delta}, \tag{13}$$

$$\text{logit}_{\text{SCALED}}(y; \alpha, x_0) := \delta \,\text{logit}\left(\Delta y + \text{logistic}(\delta^{-1}x_{\min}; \alpha, x_0)\right), \tag{14}$$

where $\delta = x_{\max} - x_{\min}$ and $\Delta = \text{logistic}(x_{\max}; \alpha, x_0) - \text{logistic}(x_{\min}; \alpha, x_0)$. As depicted in Figure 4, these scaled functions have the domain and image of interest. Scaling of the input values x by δ^{-1} makes the parameters α and x_0 comparable across vectors by making them invariant to the specific domain $[x_{\min}, x_{\max}]$ of each vector (this scaling is equivalent to using the alternative parametrization $\widetilde{\alpha} = \delta^{-1}\alpha$ and $\widetilde{x}_0 = \delta x_0$).

[3] The use of the Kumaraswamy still holds value as future embedding models may produce vectors with other distributions.

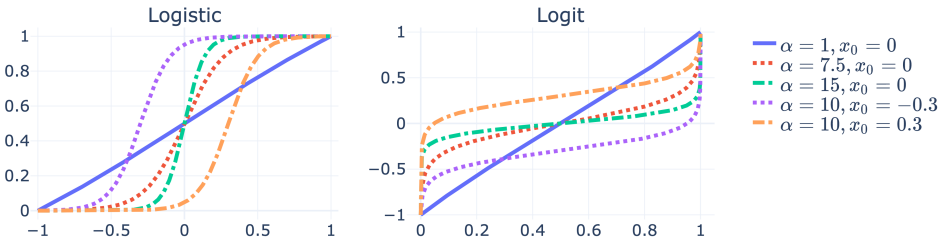

Figure 4: The scaled logistic (left) and logit (right) functions, defined in Equations (13) and (14), for different values of α and x_0 (in these examples $x_{\min} = -1$ and $x_{\max} = 1$). These nonlinearities allow to approximate bell-shaped distributions.

We solve Problem (6) with the parametrization

$$\theta := [\alpha, x_0], \qquad h := \text{logistic}_{\text{SCALED}}, \qquad h^{-1} := \text{logit}_{\text{SCALED}}\,. \tag{15}$$

It provides enough adaptability to the pair the pair Q, Q^{-1} and brings consistent improvements over a uniform quantization.

The computation is simpler than those involved when using the Kumaraswamy distribution: it only involves one exponential for h and one logarithm for h^{-1}, while h and h^{-1} involve two exponentials and two logarithms with the Kumaraswamy distribution. Moreover, only two parameters are still sufficient to characterize these nonlinearities.

For the logistic/logit nonlinearities, $\alpha \in \mathbb{R}_+$ and $x_0 \in [x_{\min}, x_{\max}]$. The projection in Line 5 of Algorithm 1 takes the form

$$\text{project}_{\mathcal{C}}(\alpha) := \max\left\{\alpha, 10^{-6}\right\}, \qquad \text{project}_{\mathcal{C}}(x_0), := \min\left\{\max\left\{x_0, x_{\min}\right\}, x_{\max}\right\}, \tag{16}$$

and we use the initial values $\boldsymbol{\mu}_{\text{init}} = [10, 0]$ and $\sigma_{\text{init}} = [2, 0.5]$.

Efficiency: The logistic and logit nonlinearities, as the Kumraswamy distribution, rely on fundamental functions (exponentials and logarithms) for their computation. However, their use is significantly reduced, as observed in Table 1, resulting in a significant acceleration.

4.3 The speed demon: Not-quite transcendental nonlinearities

The logistic and logit functions provide a good fit for vectors commonly encountered in modern ANN and a good acceleration over the Kumaraswamy distribution. However, it still relies on fundamental functions (exponentials and logarithms), which, even if used more sparingly, are still computationally expensive. We now show an alternative that bypasses fundamental functions altogether by using tight approximations.

The traditional approach to accelerate the computation of fundamental functions is using a low-degree polynomial approximation, e.g., as in Appendix A. In our application, we do not necessarily need to rely on fundamental functions as we only care about reasonable non-linearities for Problem (6). Thus, we can create our own nonlinearity instead of tightly approximating a given function. As detailed next, this leads to a significant acceleration.

Our starting point for these new nonlinearities are the base-2 logistic and logit functions

$$\text{logistic}_2(x; \alpha, x_0) := \left(1 + 2^{(-\alpha(x - x_0))}\right)^{-1}, \tag{17}$$

$$\text{logit}_2(y; \alpha, x_0) := \alpha^{-1} \log_2\left(\frac{y}{1-y}\right) + x_0. \tag{18}$$

Qualitatively, these functions behave exactly like their natural counterparts. However, the binary base is useful for dealing with floating-point numbers. A positive floating-point number z is internally represented as $z = m \cdot 2^p$, where $m \in [0.5, 1)$ is the mantissa and the integer p is the exponent. Then, to implement logit_2 we can use the identity

$$\log_2(z) = \log_2(m) + p. \tag{19}$$

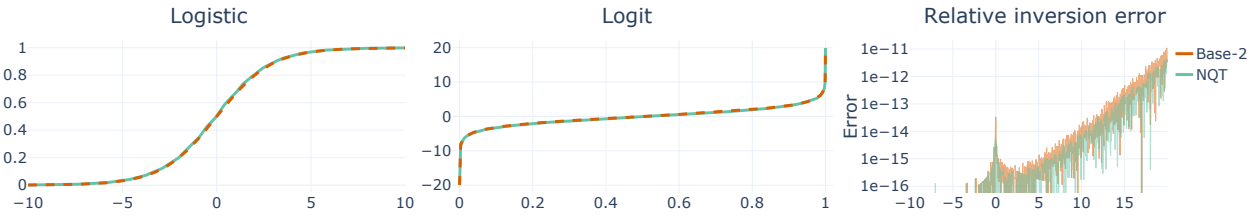

Figure 5: The NQT logistic and logit functions in Equations (21) and (22) closely follow their standard base-2 counterparts in Equations (17) and (18). The differences are only noticeable when zooming in. Moreover, the NQT logit is the inverse of the NQT logistic with a numerical accuracy on par with the base-2 counterparts.

The not-quite transcendental (NQT) function [37] is a piecewise-linear interpolation of $\log_2$ [23], defined as

$$\log_{\text{NQT}}(z) := 2(m-1) + p. \tag{20}$$

With these elements, we can introduce our new nonlinearities.

Definition 1. *Let $y \in (0,1)$, $\alpha \in \mathbb{R}_+$, and $x_0 \in \mathbb{R}$. Let $m \cdot 2^p$ be the binary representation of $z = y/(1-y)$. We define the NQT logit as*

$$\text{logit}_{\text{NQT}}(y; \alpha, x_0) := \alpha^{-1}\left[2(m-1) + p\right] + x_0. \tag{21}$$

The NQT logit is continuous, piecewise differentiable, monotonically increasing, and invertible. Using simple manipulations, we derive its inverse as defined next.

Definition 2. *Let $x \in \mathbb{R}$, $\alpha \in \mathbb{R}_+$, and $x_0 \in \mathbb{R}$. The NQT logistic function, as*

$$\text{logistic}_{\text{NQT}}(x; \alpha, x_0) := \frac{m \cdot 2^p}{m \cdot 2^p + 1} \quad \textit{where} \quad \begin{aligned} p &= \lfloor \alpha(x - x_0) + 1 \rfloor, \\ m &= \frac{\alpha(x - x_0) - p}{2} + 1. \end{aligned} \tag{22}$$

For our NQT-based quantization, we define $\text{logistic}_{\text{NQT,SCALED}}$ and $\text{logit}_{\text{NQT,SCALED}}$, the scaled versions of $\text{logistic}_{\text{NQT}}$ and $\text{logit}_{\text{NQT}}$ exactly as in Equations (13) and (14); we omit the equations for brevity. We solve Problem (6) with the parametrization

$$\theta := [\alpha, x_0], \qquad h := \text{logistic}_{\text{NQT,SCALED}}, \qquad h^{-1} := \text{logit}_{\text{NQT,SCALED}}. \tag{23}$$

The NQT nonlinearities share the parameters $\alpha \in \mathbb{R}_+$ and $x_0 \in [x_{\min}, x_{\max}]$ with the logistic/logit nonlinearities. We therefore use the projections in Equation (16) in Line 5 of Algorithm 1 and the same initial values $\boldsymbol{\mu}_{\text{init}} = [10, 0]$ and $\sigma_{\text{init}} = [2, 0.5]$.

Efficiency: The NQT nonlinearities, in contrast to the logistic/logit functions, do not rely on fundamental functions (exponentials and logarithms) for their computation and can be implemented with a handful of elementary operations, see Table 1. Implementation details are provided in Appendix B.

5 Evaluation

We now evaluate the ability of NVQ to provide computationally-efficient low-loss quantization. First, we examine how effectively NVQ can reduce loss versus the uniform quantization baseline. Next, we assess the utility of using per-vector quantizers. We then examine the effect that both non-linearity selection and subvector count have on loss. Finally, we evaluate metrics that are important in production deployment, including query recall versus latency and storage footprint savings.

Throughout the experiments, we use the shorthand Log-Log and NQT to refer to the logistic/logit and NQT logistic/logit nonlinearities, respectively. Before quantizing, we center the data using the mean vector of the dataset $\mathcal{X}$, i.e., we subtract the vector $\bar{\mathbf{x}} = \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}$. This type of centering has proven effective and robust to distribution shifts [2]. We use the datasets described in Table 2, which stem from different embedding models and types of data.

[4] https://github.com/datastax/jvector

Table 2: Datasets used for our experimental results. We denote the dimensionality and the number of vectors in the dataset by d and n, respectively.

Name	d	n
ada-002-100k[4]	1536	10^5
openai-v3-100k[4]	1536	10^5
gecko-100k[4]	768	10^5

Name	d	n
dpr-1M [2]	768	10^6
cohere-1M[5]	1024	10^6
cap-1M[6]	1536	10^6

Name	d	n
dpr-10M [2]	768	10^7
cohere-10M[5]	1024	10^7
cap-6M[6]	1536	$6.07 \cdot 10^6$

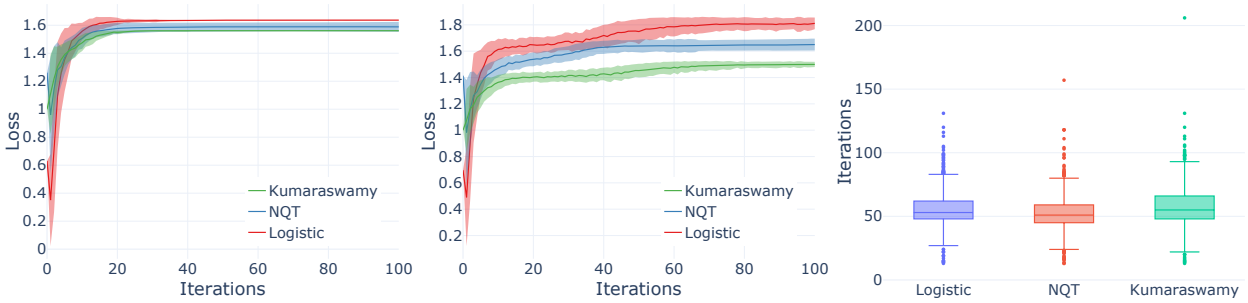

Figure 6: **Left and center:** Evolution of the objective function (higher is better) in Problem (6) for different runs of Algorithm 1 (the shaded region indicates plus/minus one standard deviation due to the stochasticity of the natural gradient) for 4-bit (left) and 8-bit (right) NVQ. **Right:** The number of iterations of Algorithm 1 to reach the stopping condition in Section 3.1 (i.e., an absolute error of 10^{-4} across consecutive iterations) with $\beta = 8$ bits for vectors in ada002-100k. On average, 50 iterations suffice.

5.1 Loss reduction and convergence

We start by examining the capability of Algorithm 1 to optimize Problem (6). In the left and center plots of Figure 6, we analyze the convergence of several runs of Algorithm 1 for a single vector. For this, we remove the stopping condition in Section 3.1 and run the algorithm for a fixed number of 100 iterations. Since SNES uses a stochastic estimate of the natural gradient, we observe relatively small variability across different runs of the algorithm. All runs are successful and achieve an improvement over the baseline (recall that the objective function in Problem (6) is the relative improvement over the uniform baseline). Few iterations are required to reach convergence. After 20 iterations, the algorithm converges for 4 bits and minor improvements are achieved for 8 bits (no noticeable change after 50 iterations). This is corroborated in right plot, where we observe that 50 iterations are sufficient on average (computed across vectors) to reach an absolute error of 10^{-4} between two consecutive iterations.

5.2 Value of per-vector quantizers

In Figure 7 we depict the solutions found by Algorithm 1 when optimizing Problem (6) for many vectors. Different parameters are chosen for different vectors. This sizable spread in the selected parameters gives grounds for the per-vector approach in NVQ. Additionally, the Log-Log and NQT solutions, being similar in nature, exhibit similar parameter distributions (shifted by the change from the natural base to base 2).

In Figure 8 (left), we can observe that each solution found by Algorithm 1 for different vectors (see Figure 7) is better than what can be achieved with uniform quantization (i.e., no solution is below 1). The Kumaraswamy, Log-Log, and NQT nonlinearities produce improvements of ~1.81x, ~1.90x, and ~1.72x on average, respectively, while reaching ~4x for some vectors.

[5] https://huggingface.co/datasets/Cohere/wikipedia-22-12
[6] https://huggingface.co/datasets/laion/Caselaw_Access_Project_embeddings

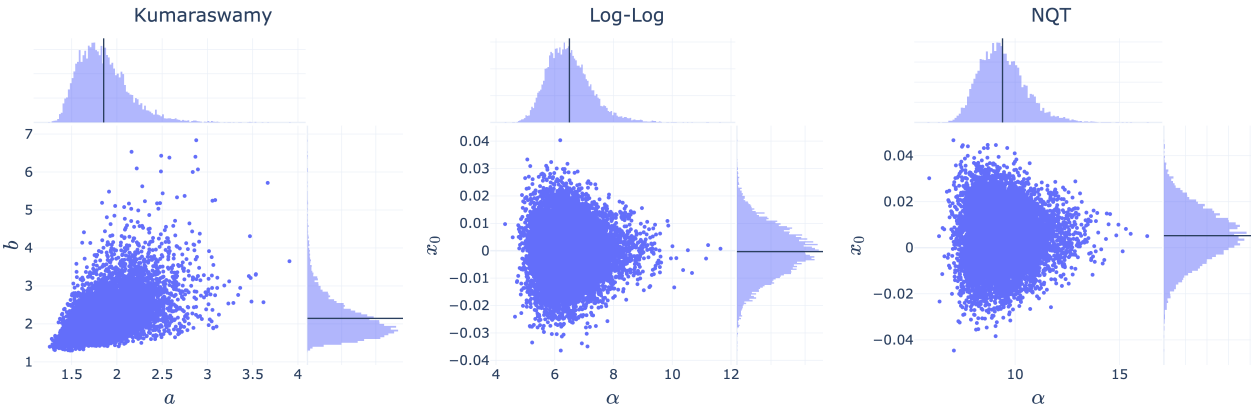

Figure 7: The 8-bit NVQ solutions for 10^4 vectors (each circle represents the solution for one vector) from ada002-100k. Different nonlinearity parameters are chosen for each vector, justifying the individualized per-vector learning. This pattern is highly repeatable over datasets, as obhserved in Figure 15 of the appendix.

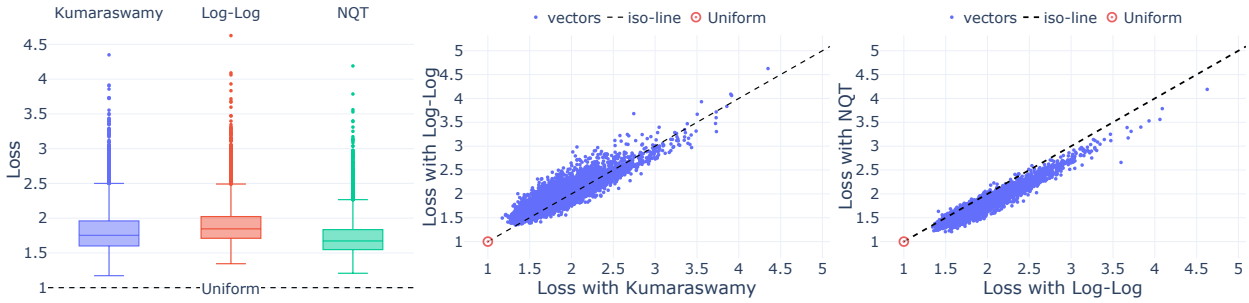

Figure 8: **(Left)** The objective function values achieved when maximizing Problem (6) with $\beta = 8$ bits for 10^4 vectors from ada002-100k (the objective function is the relative improvement over the uniform scalar quantization). On average, we achieve an improvement of ~1.8x (higher is better). There are no vectors that see no improvement. **(Center and right)** Comparison of the objective function values achieved when optimizing Problem (6) with $\beta = 8$ bits using different nonlinearities for 10^4 vectors from ada002-100k (each vector is represented by a circle). At the iso-line, both nonlinearities are equally good. Log-Log is slightly better for most vectors compared to the Kumaraswamy (center) and NQT (right) nonlinearities.

5.3 Comparison of non-linearities

We also individually compare the objective function values of the solutions found with different nonlinearities in Figure 8 (center and right). In general, Log-Log is slightly better than the Kumaraswamy and NQT nonlinearities for vectors stemming for commonly used embedding models. However, as usual in machine learning, there is no silver bullet: the NQT nonlinearity is faster to compute but the Kumaraswamy nonlinearity offers more representational flexibility.

5.4 Benefit of subvectors

We study the effects of changing the number of subvectors in Figure 9. We observe that using more subvectors improves the value of the objective function in Problem (6). This is natural as the domain $[x_{\min}, x_{\max}]$ corresponmdig to each subvector becomes tighter as the number of subvectors increases. This effect is clearly reflected in other vector search metrics, as depicted in the bottom row of Figure 9. These improvements come with a slightly increase of the payload of each compressed vector, as described in Section 3.2.

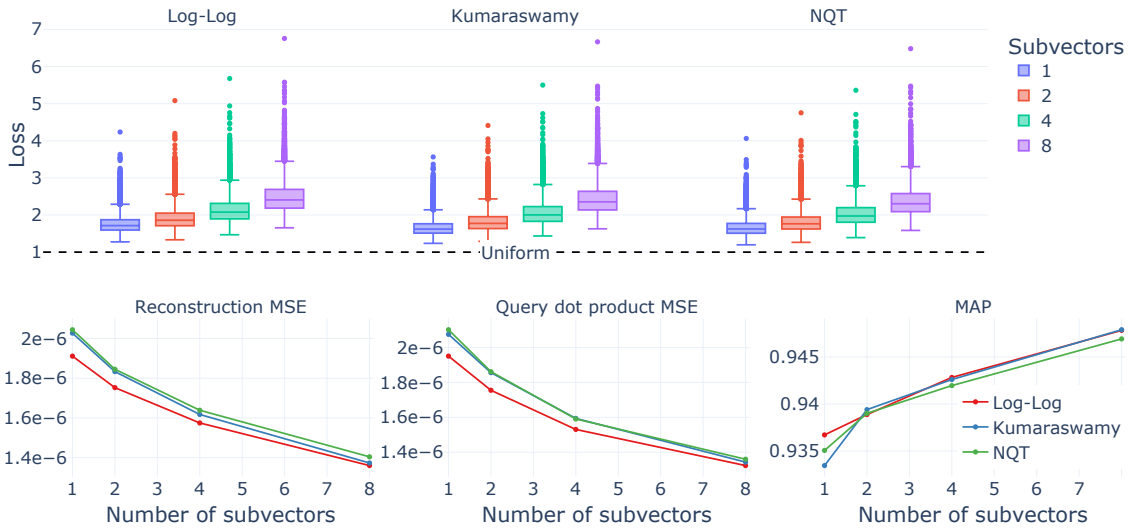

Figure 9: (**Top**) Increasing the number of subvectors increases the objective function (higher is better) in Problem (6). There are no vectors that see no improvement over the baseline. (**Bottom**) The average reconstruction error and the query dot product error, defined as $\sum_{\mathbf{x}\in\mathcal{X}}(\langle\mathbf{q},\mathbf{x}\rangle - \langle\mathbf{q},\tilde{\mathbf{x}}\rangle)^2$ decrease (lower is better) and the mean average precision (MAP) increases (higher is better). Results obtained with $\beta = 4$ for 10^4 vectors from ada002-100k. Other datasets and values of β are in Figures 16 and 17 of the appendix.

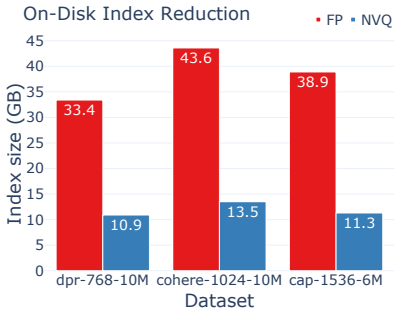

Figure 10: The on-disk vector index size for FP32 vectors (red) and NVQ vectors (blue), for larger datasets of 768, 1024, and 1536 dimensions. In thse cases, NVQ achieves storage savings of 3.06x, 3.23x, and 3.44x with a minute loss in accuracy (see Figure 11).

5.5 Evaluation of search storage footprint

As described in Section 2, vector quantization is commonly used to conserve memory and accelerate search at the expense of recall. Reranking search results with full-precision vectors can mitigate the impact but requires random reads from storage of a small multiple of k vectors for top-k retrieval. For a production system servicing thousands of queries/sec, the storage costs are driven by the amount of high-performance capacity required, which can be expensive for large-scale datasets, making this the 3rd largest system cost after compute and memory.

We measure the impact that NVQ has on the index storage capacity requirements in Figure 10. Shown is the storage footprint in terms of measured file size for datasets encoded using 8-bit NVQ with two subvectors versus the original FP32 encoding. NVQ uses 3.06x, 3.23x, and 3.44x less storage than FP32 for the 768-, 1024-, and 1536-dimensional datasets, respectively. The savings are higher for higher dimensionality due to the reduced format overhead.

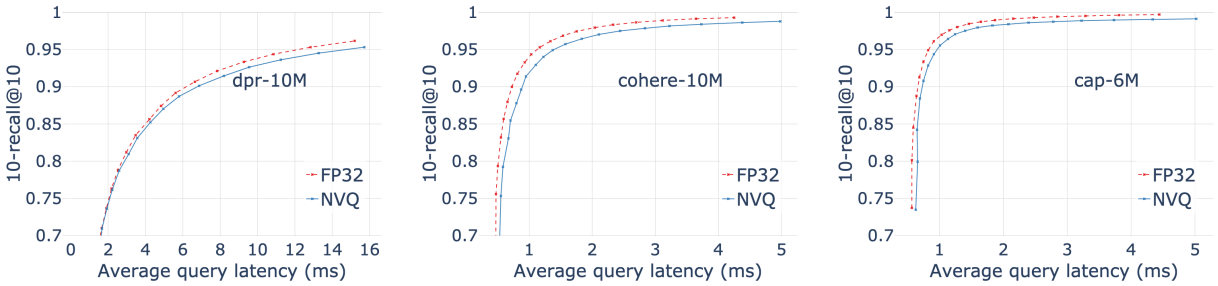

Figure 11: Recall versus query latency for three larger datasets. A graph-based index was constructed using PQ vectors. Reranking was then performed using either NVQ or FP32 vectors. The results show that NVQ has a very small deleterious effect on recall in the high-recall regime for 768 dimensions **(Left)**, 1024 dimensions **(Center)**, and 1536 dimensions **(Right)**.

5.6 Evaluation of search speed

Finally, we look at search recall versus latency on several larger datasets with a range of dimensionality from 768 to 1536, a popular range for today's retrieval-augmented generation systems. This evaluation exposes the impact of NVQ reconstruction loss in terms of degraded recall as well as the cost of decoding the quantized vectors in terms of latency. A graph index was constructed using the Vamana algorithm described in [46] (out-degree $R = 64$, search window size of 200, $\alpha = 1.2$) and vectors encoded using Product Quantization (256 centroids, 16x compression). Queries were then performed from a held-out query set of 10k queries while the amount of reranking was swept from 1.0x (i.e., top-k PQ vectors reranked with k FP or NVQ vectors) to a sufficiently-high ratio (e.g., 40-80x). In the high-recall regime of most interest (> 0.95), 10-recall@10 is impacted less than 0.01 at the same latency.

6 Conclusion

We introduced Non-uniform Vector Quantization (NVQ) for high-fidelity compression of vectors for similarity search at modern embedding dimensionalities. The key idea is to replace a one-size-fits-all uniform scalar quantizer with an *individualized*, two-parameter, non-uniform quantizer per vector (or subvector), chosen by a tiny optimization problem at insert time. We showed that three parsimonious nonlinearities- Kumaraswamy, scaled logistic/logit, and non-quite-transcendental logistic/logit- span a useful representational-speed Pareto front, and that a lightweight, gradient-free optimizer reliably finds good settings in a few dozen iterations. Empirically in both 4- and 8-bit regimes, NVQ consistently reduces reconstruction loss relative to uniform scalar quantization by 1.7 to 1.9x, reaching 4x for some vectors. Subvectorization into 2,4, or 8 subvectors further improves reconstruction with minimal overhead. When NVQ is used to quantize vectors for reranking, NVQ can reduce storage use by 3.06 to 3.44x versus 32-bit full-precision vectors for 768 to 1536 dimensional data, respectively, with an impact of less than 0.01 on recall above 0.95.

NVQ also has limitations that point to promising directions. First, we optimized mean-squared reconstruction error as a surrogate for ranking quality; incorporating query-aware or rank-based losses could further tighten recall–latency trade-offs (e.g., see [47]). Second, our normalization relies on per-(sub)vector min/max and may be sensitive to outliers; robust or learned normalizers are worth exploring. Third, while we evaluated 4-8 bit data types, pushing into ultra-low bitrates will likely require additional structure (e.g., mixed precision across dimensions) or multi-level schemes. Finally, avoiding full dequantization during reranking by deriving unbiased dot-product estimators in the quantized domain (in the spirit of the asymmetric distance computation from [26]) could reduce compute and memory traffic further, especially on accelerators.

References

[1] Cecilia Aguerrebere et al. *Locally-Adaptive Quantization for Streaming Vector Search*. arXiv:2402.02044 [cs]. Feb. 2024.

[2] Cecilia Aguerrebere et al. "Similarity Search in the Blink of an Eye with Compressed Indices". In: *Proc. VLDB Endow.* 16.11 (July 2023), pp. 3433–3446.

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. "Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan". In: *Proceedings of the VLDB Endowment* 9.4 (Dec. 2015), pp. 288–299.

[4] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. "Quicker ADC : Unlocking the Hidden Potential of Product Quantization With SIMD". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.5 (May 2021), pp. 1666–1677.

[5] Sunil Arya and David M. Mount. "Approximate nearest neighbor queries in fixed dimensions". In: *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*. USA: Society for Industrial and Applied Mathematics, Jan. 1993, pp. 271–280.

[6] Martin Aumüller and Matteo Ceccarello. "Recent Approaches and Trends in Approximate Nearest Neighbor Search". In: *IEEE Data Engineering Bulletin* (2023).

[7] Artem Babenko and Victor Lempitsky. "Additive Quantization for Extreme Vector Compression". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 931–938.

[8] Dmitry Baranchuk et al. "DeDrift: Robust Similarity Search under Content Drift". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 10992–11001.

[9] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*. arXiv:1308.3432. Aug. 2013.

[10] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517.

[11] Andreas Blattmann et al. "Retrieval-Augmented Diffusion Models". en. In: *Advances in Neural Information Processing Systems* 35 (Dec. 2022), pp. 15309–15324.

[12] Sebastian Borgeaud et al. "Improving Language Models by Retrieving from Trillions of Tokens". en. In: *Proceedings of the 39th International Conference on Machine Learning*. PMLR, June 2022, pp. 2206–2240.

[13] Lawrence Cayton. "Fast nearest neighbor retrieval for Bregman divergences". In: *Proceedings of the 25th international conference on Machine learning*. New York, NY, USA: Association for Computing Machinery, July 2008, pp. 112–119.

[14] Qi Chen et al. "SPANN: highly-efficient billion-scale approximate nearest neighbor search". In: *Proceedings of the 35th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 5199–5212.

[15] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186.

[16] Magdalen Dobson et al. "Scaling Graph-Based ANNS Algorithms to Billion-Size Datasets: A Comparative Analysis". In: *CoRR* abs/2305.04359 (2023). arXiv: 2305.04359.

[17] Cong Fu, Changxu Wang, and Deng Cai. "High dimensional similarity search with satellite system graph: efficiency, scalability, and unindexed query compatibility". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.8 (Aug. 2022), pp. 4139–4150.

[18] Cong Fu et al. "Fast approximate nearest neighbor search with the navigating spreading-out graph". In: *Proceedings of the VLDB Endowment* 12.5 (Jan. 2019), pp. 461–474.

[19] Tiezheng Ge et al. "Optimized Product Quantization for Approximate Nearest Neighbor Search". In: *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*. USA: IEEE Computer Society, June 2013, pp. 2946–2953.

[20] Yunchao Gong et al. "Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (Dec. 2013), pp. 2916–2929.

[21] Mihajlo Grbovic et al. "Scalable Semantic Matching of Queries to Ads in Sponsored Search Advertising". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, July 2016, pp. 375–384.

[22] Ruiqi Guo et al. "Accelerating large-scale inference with anisotropic vector quantization". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. JMLR.org, July 2020, pp. 3887–3896.

[23] E.L. Hall, D.D. Lynch, and S.J. Dwyer. "Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications". In: *IEEE Transactions on Computers* C-19.2 (Feb. 1970). Conference Name: IEEE Transactions on Computers, pp. 97–105.

[24] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. New York, NY, USA: Association for Computing Machinery, May 1998, pp. 604–613.

[25] Omid Jafari et al. *A Survey on Locality Sensitive Hashing Algorithms and their Applications*. arXiv:2102.08942 [cs]. Feb. 2021.

[26] Herve Jégou, Matthijs Douze, and Cordelia Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (Jan. 2011), pp. 117–128.

[27] Yanrong Ji et al. "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome". In: *Bioinformatics* 37.15 (Aug. 2021), pp. 2112–2120.

[28] M. C. Jones. "Kumaraswamy's distribution: A beta-type distribution with some tractability advantages". In: *Statistical Methodology* 6.1 (Jan. 2009), pp. 70–81.

[29] Vladimir Karpukhin et al. "Dense Passage Retrieval for Open-Domain Question Answering". In: *Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2020, pp. 6769–6781.

[30] Anthony Ko et al. *Low-Precision Quantization for Efficient Nearest Neighbor Search*. arXiv:2110.08919 [cs]. Oct. 2021.

[31] Ponnambalam Kumaraswamy. "A generalized probability density function for double-bounded random processes". In: *Journal of Hydrology* 46.1 (Mar. 1980), pp. 79–88.

[32] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive NLP tasks". In: *Advances in Neural Information Processing Systems*. Curran Associates Inc., Dec. 2020, pp. 9459–9474.

[33] Wen Li et al. "Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement". In: *IEEE Transactions on Knowledge and Data Engineering* 32.8 (Aug. 2020), pp. 1475–1488.

[34] Yujia Li et al. "Competition-level code generation with AlphaCode". In: *Science* 378.6624 (Dec. 2022). Publisher: American Association for the Advancement of Science, pp. 1092–1097.

[35] Defu Lian et al. "LightRec: A Memory and Search-Efficient Recommender System". In: *Proceedings of The Web Conference 2020*. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 695–705.

[36] Yu A. Malkov and D. A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (Apr. 2020), pp. 824–836.

[37] Jonah M. Miller, Joshua C. Dolence, and Daniel Holladay. *Not-Quite Transcendental Functions and their Applications*. arXiv:2206.08957. June 2022.

[38] Marius Muja and David G. Lowe. "Scalable Nearest Neighbor Algorithms for High Dimensional Data". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (Nov. 2014), pp. 2227–2240.

[39] James Jie Pan, Jianguo Wang, and Guoliang Li. "Survey of vector database management systems". In: *The VLDB Journal* 33.5 (July 2024), pp. 1591–1615.

[40] John Paparrizos et al. "Fast Adaptive Similarity Search through Variance-Aware Quantization". In: *2022 IEEE 38th International Conference on Data Engineering*. Kuala Lumpur, Malaysia: IEEE, May 2022, pp. 2969–2983.

[41] Yun Peng et al. "Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases". In: *Proc. ACM Manag. Data* 1.1 (May 2023), 54:1–54:27.

[42] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". en. In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, July 2021, pp. 8748–8763.

[43] Nina Shvetsova et al. "Everything at Once - Multi-Modal Fusion Transformer for Video Retrieval". en. In: 2022, pp. 20020–20029.

[44] Chanop Silpa-Anan and Richard Hartley. "Optimised KD-trees for fast image descriptor matching". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. June 2008, pp. 1–8.

[45] Sadagopan Srinivasan et al. "CMP memory modeling: How much does accuracy matter?" In: *Workshop on Modeling, Benchmarking and Simulation*. 2009.

[46] Suhas Jayaram Subramanya et al. "DiskANN: fast accurate billion-point nearest neighbor search on a single node". In: *Advances on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 13766–13776.

[47] Mariano Tepper et al. "LeanVec: Searching vectors faster by making them fit". en. In: *Transactions on Machine Learning Research* (Jan. 2024).

[48] Karthik V. et al. "BANG: Billion-Scale Approximate Nearest Neighbor Search using a Single GPU". In: (2024). Publisher: arXiv Version Number: 4.

[49] Mohammad Hassan Vali and Tom Bäckström. "NSVQ: Noise Substitution in Vector Quantization for Machine Learning". In: *IEEE Access* 10 (2022), pp. 13598–13610.

[50] Jingdong Wang et al. "A Survey on Learning to Hash". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (Apr. 2018), pp. 769–790.

[51] Mengzhao Wang et al. "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search". In: *Proceedings of the VLDB Endowment* 14.11 (July 2021), pp. 1964–1978.

[52] Runhui Wang and Dong Deng. "DeltaPQ: lossless product quantization code compression for high dimensional similarity search". In: *Proc. VLDB Endow.* 13.13 (Sept. 2020), pp. 3603–3616.

[53] Zeyu Wang et al. "Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions". en. In: *IEEE Data Engineering Bulletin* (Jan. 2023).

[54] Max Wasserman and Gonzalo Mateos. *Stabilizing the Kumaraswamy Distribution*. arXiv:2410.00660 [cs]. Oct. 2024.

[55] Daan Wierstra et al. "Natural Evolution Strategies". In: *Journal of Machine Learning Research* 15.27 (2014), pp. 949–980.

[56] Zhi Xu et al. "Residual Vector Product Quantization for Approximate Nearest Neighbor Search". In: *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference*. Berlin, Heidelberg: Springer-Verlag, May 2022, pp. 208–220.

[57] Ting Zhang, Chao Du, and Jingdong Wang. "Composite Quantization for Approximate Nearest Neighbor Search". en. In: *Proceedings of the 31st International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, June 2014, pp. 838–846.

A Fast logarithms and exponentiation

Even if the expressions of F_{KS} and F_{KS}^{-1} are mathematically straightforward, the use of fundamental functions makes their computation challenging. To speedup these computations, we use the identity

$$x^c = \exp(c \log x), \tag{24}$$

enabling the use of fast approximations of the common functions exp and log for $x \in \mathbb{R}_+$.

For the exponential and logarithm, we use an implementation based on the minimax polynomial approximation.[7] We have modified the Remez polynomials for an optimal tradeoff between speed and accuracy using https://github.com/DKenefake/OptimalPoly/. We reproduce the code for completeness.

```
#include <cmath>
#include <cstdint>
#include <cstring>

float int_as_float (int32_t a) {
    float r;
    memcpy (&r, &a, sizeof r);
    return r;
}
int32_t float_as_int (float a) {
    int32_t r;
    memcpy (&r, &a, sizeof r);
    return r;
}

float fast_logf (float a) {
    float m, r, s, t, i, f;
    int32_t e;

    e = (float_as_int(a) - 0x3f2aaaab) & 0xff800000;
    m = int_as_float(float_as_int(a) - e);
    i = (float) e * 1.19209290e-7f;
    f = m - 1.0f;
    s = f * f;
    r = fmaf(0.230836749f, f, -0.279208571f);
    t = fmaf(0.331826031f, f, -0.498910338f);
    r = fmaf(r, s, t);
    r = fmaf(r, s, f);
    r = fmaf(i, 0.693147182f, r);
    return r;
}

float fast_exp(float x) {
    float invlog2e = 1.442695041f;  // 1 / log2(e)
    float expCvt = 12582912.0f;  // 1.5 * (1 << 23)

    /* exp(x) = 2^i * 2^f; i = rint (log2(e) * x), -0.5 <= f <= 0.5 */
    float t = x * invlog2e;  // t = x / log2(e)
    float r = (t + expCvt) - expCvt;  // r = round(t)
    float f = t - r;  // f = t - round(t)
    int i = (int) r; // i = (int) r
```

[7] https://stackoverflow.com/a/39822314 and https://stackoverflow.com/a/47025627

```
    float temp = fmaf(f, 0.009651907610706037f, 0.05593479631997887f);
    temp = fmaf(temp, f, 0.2402301551437674f);
    temp = fmaf(temp, f, 0.6931186232012877f);
    temp = fmaf(temp, f, 0.9999993887682104f);

    temp = int_as_float(float_as_int(temp) + (i << 23));  // temp = temp * 2^i
    return temp;
}
```

B Fast NQT logistic and logit functions

```
#include <cmath>
#include <cstdint>
#include <cstring>

float int_as_float (int32_t a) {
    float r;
    memcpy (&r, &a, sizeof r);
    return r;
}
int32_t float_as_int (float a) {
    int32_t r;
    memcpy (&r, &a, sizeof r);
    return r;
}

float logisticNQT(float value, float alpha, float x0) {
    // The value -alpha * x0 can be precomputed
    float temp = fmaf(value, alpha, -alpha * x0);
    int p = round(temp + 0.5f);
    int m = float_as_int(fmaf(temp - p, 0.5f, 1f));

    temp = int_as_float(m + (p << 23));  // temp = m * 2^p
    return temp / (temp + 1f);
}

float logitNQT(float value, float inverseAlpha, float x0) {
    float z = value / (1f - value);

    int temp = float_as_int(z);
    int e = temp & 0x7f800000;
    float p = (float) ((e >> 23) - 128);
    float m = int_as_float((temp & 0x007fffff) + 0x3f800000);

    return fmaf(m + p, inverseAlpha, x0);
}
```

C Additional experimental results

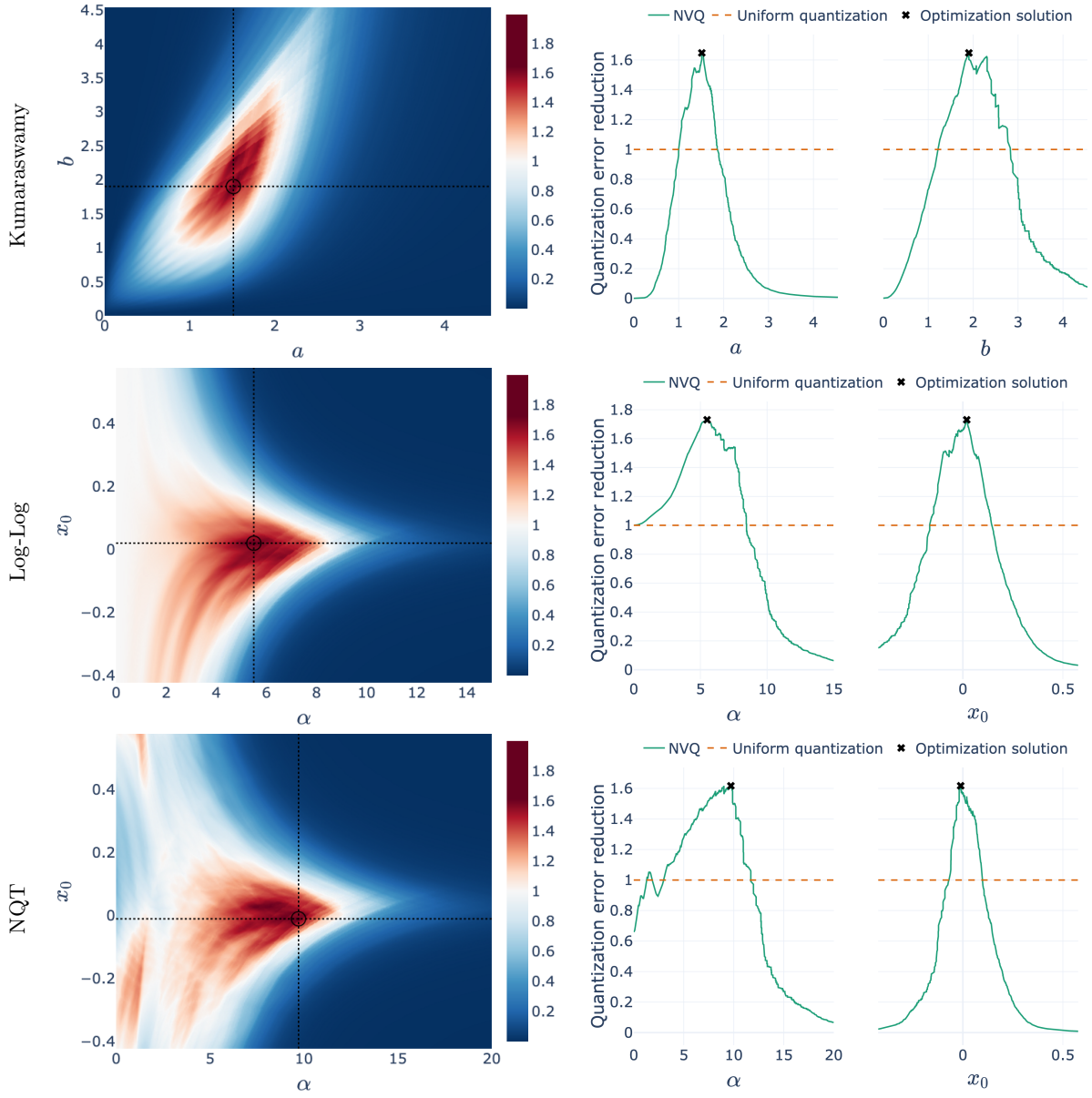

Figure 12: Algorithm 1 finds a good maximum of Problem (6) using the nonlinearities presented in Section 4 for $\beta = 4$ bits. **Left:** the landscape of the objective function in Problem (6), which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), as a function of the nonlinearity parameters for one vector from ada002-100k (see Table 2). The solution found by Algorithm 1 is marked by a black circle. **Right:** two cross cuts taken at the values corresponding to the found solution.

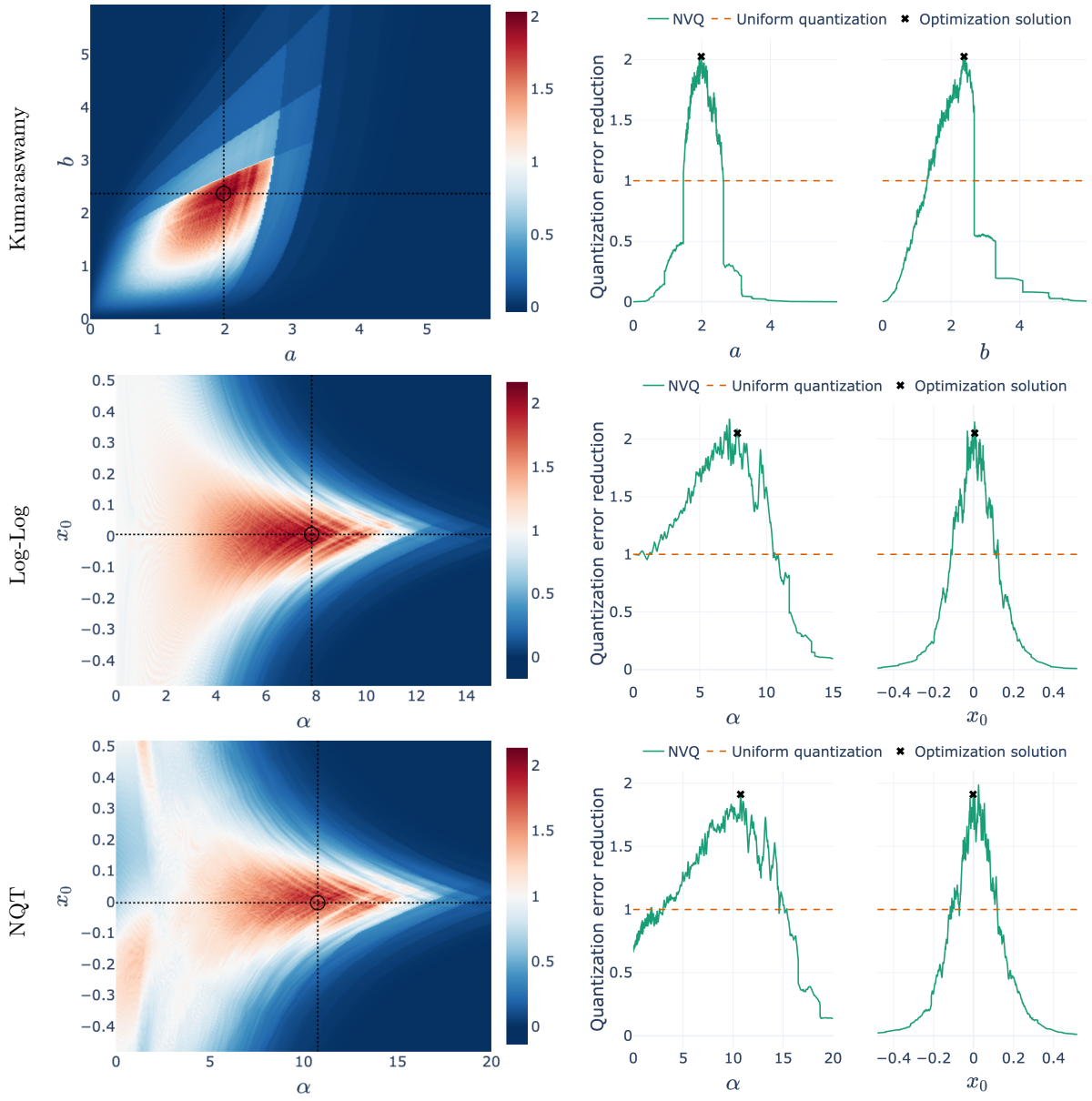

Figure 13: Algorithm 1 finds a good maximum of Problem (6) using the nonlinearities presented in Section 4 for $\beta = 8$ bits. **Left:** the landscape of the objective function in Problem (6), which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), as a function of the nonlinearity parameters for one vector from openai-v3-100k . The solution found by Algorithm 1 is marked by a black circle. **Right:** two cross cuts taken at the values corresponding to the found solution.

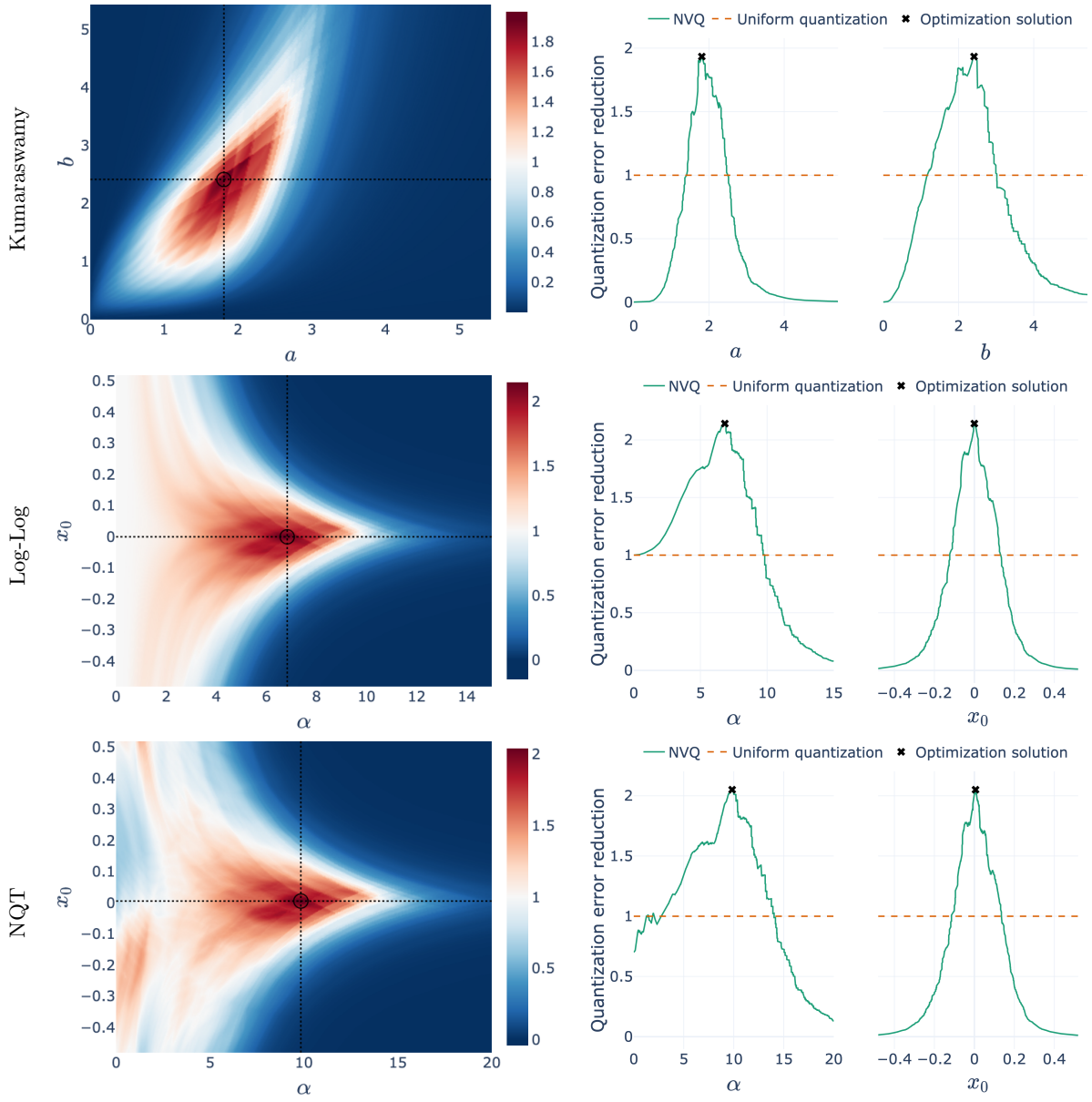

Figure 14: Algorithm 1 finds a good maximum of Problem (6) using the nonlinearities presented in Section 4 for $\beta = 4$ bits. **Left:** the landscape of the objective function in Problem (6), which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), as a function of the nonlinearity parameters for one vector from openai-v3-100k . The solution found by Algorithm 1 is marked by a black circle. **Right:** two cross cuts taken at the values corresponding to the found solution.

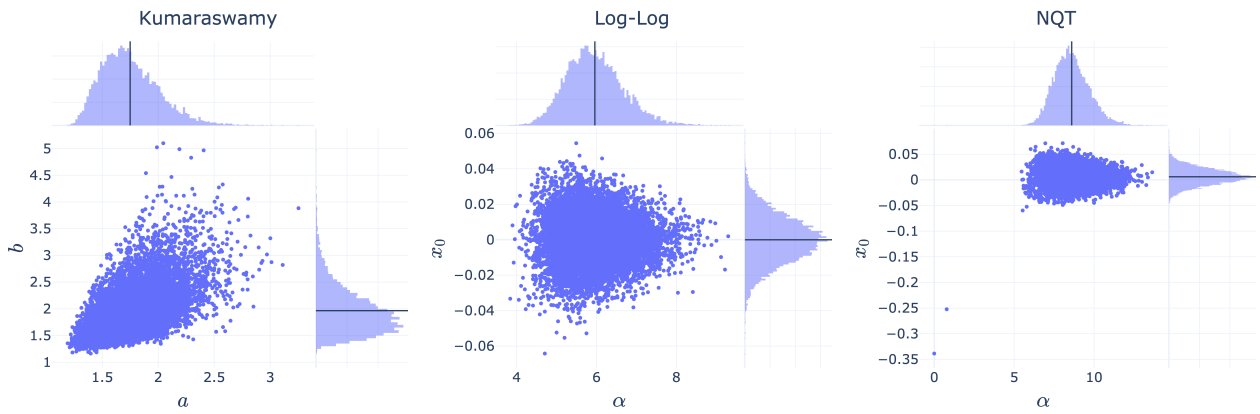

Figure 15: The solutions for 10^4 vectors from gecko-100k. Different nonlinearity parameters are chosen for each vector.

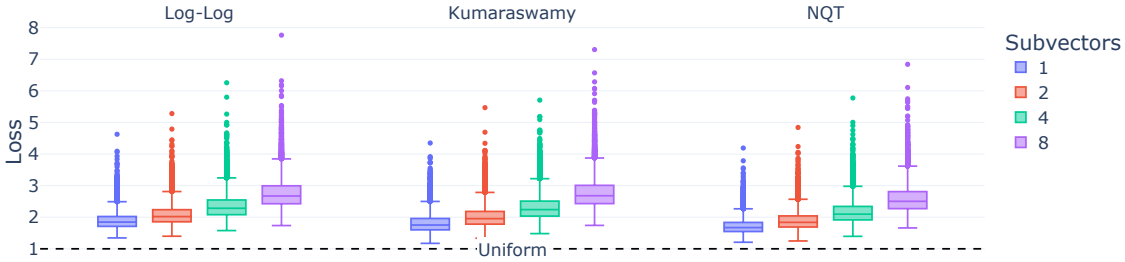

Figure 16: Increasing the number of subvectors (depicted here for $\beta = 8$) increases the objective function (higher is better) in Problem (6) for 10^4 vectors from ada002-100k.

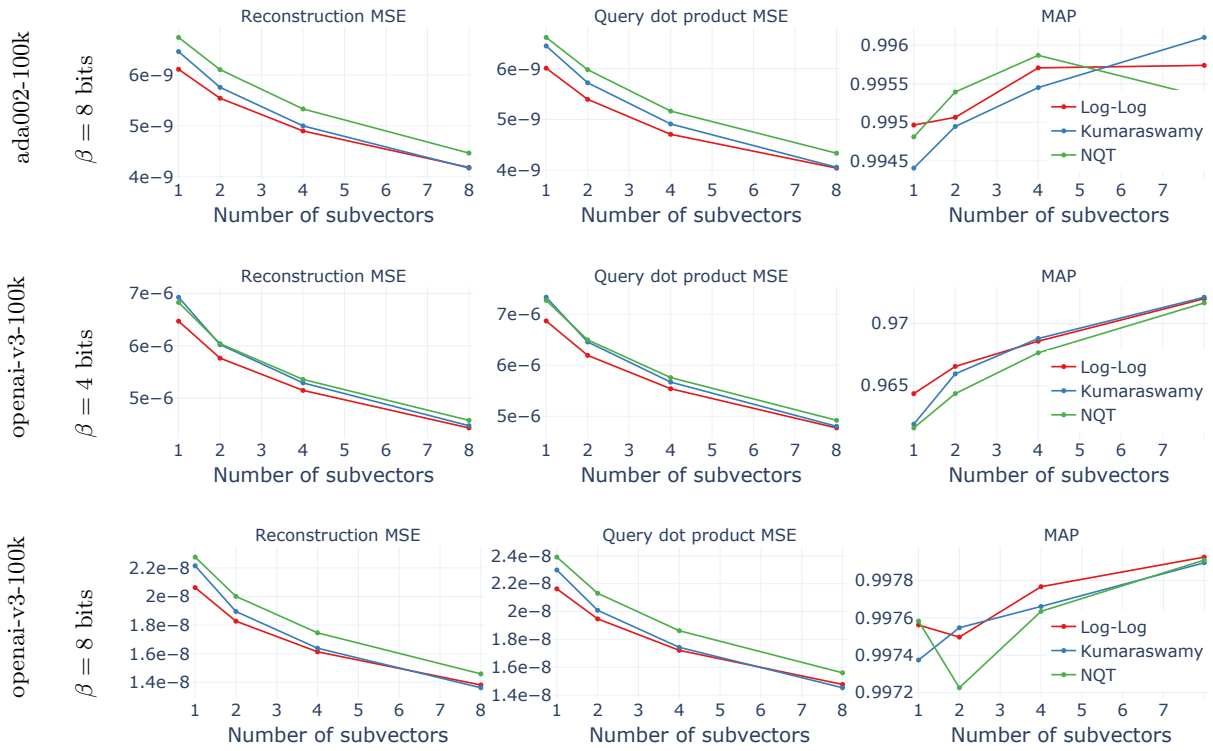

Figure 17: Increasing the number of subvectors lowers the average reconstruction error and the query dot product error, defined as $\sum_{\mathbf{x} \in \mathcal{X}} (\langle \mathbf{q}, \mathbf{x} \rangle - \langle \mathbf{q}, \widetilde{\mathbf{x}} \rangle)^2$, and increases the mean average precision (MAP) in openai-v3-100k.