Supplemental Material for "Individualized non-uniform quantization for vector search"

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

$$x^c = \exp(c \log x), \tag{25}$$

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.[1] 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);
```

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

```
    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

    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

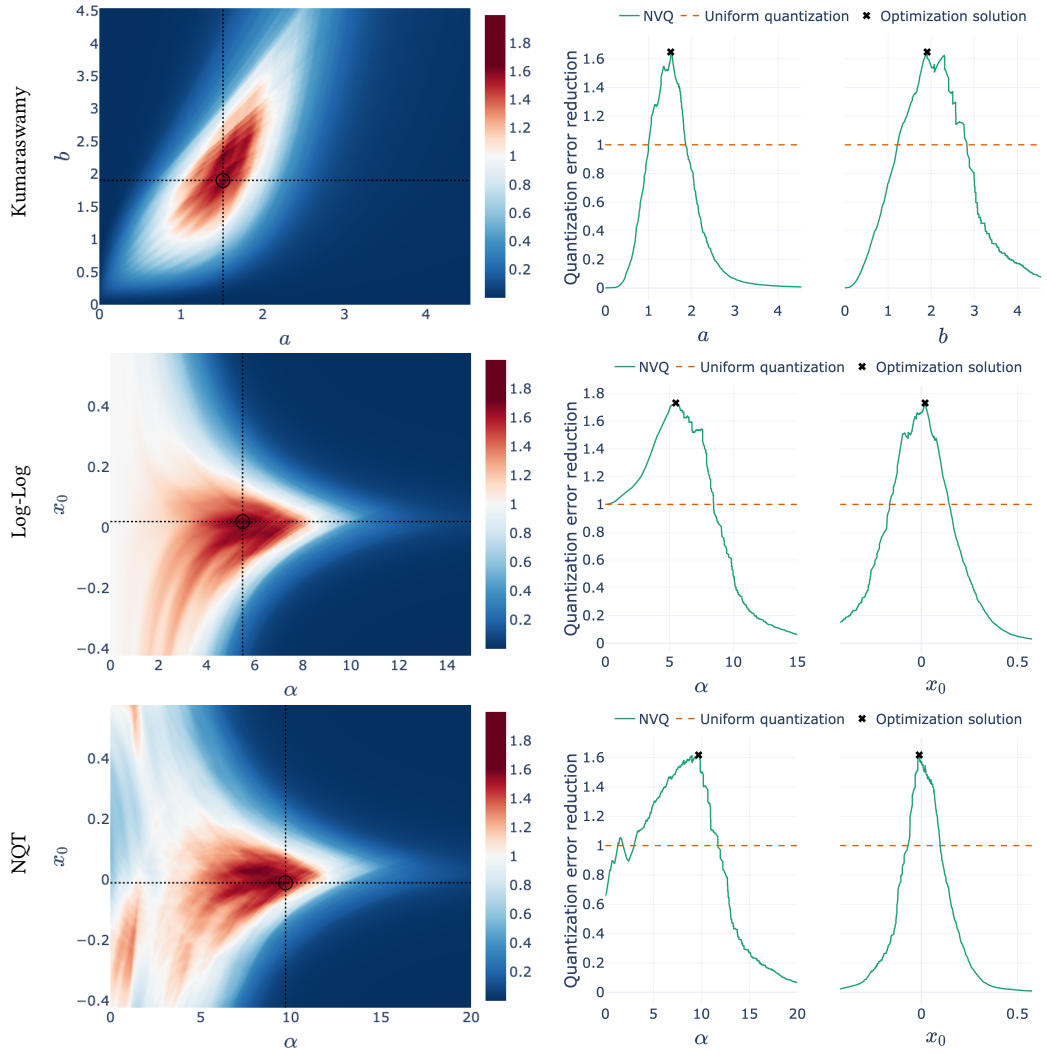

Fig. 12. Algorithm 1 finds a good maximum of Problem 6 using the nonlinearities presented in Section 4 for $\beta = 4$ bits. **Left:** the landscape of the objective function in Problem 6, which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), as a function of the nonlinearity parameters for one vector from ada002-100k (see Table 2). The solution found by Algorithm1 is marked by a black circle. **Right:** two cross cuts taken at the values corresponding to the found solution.

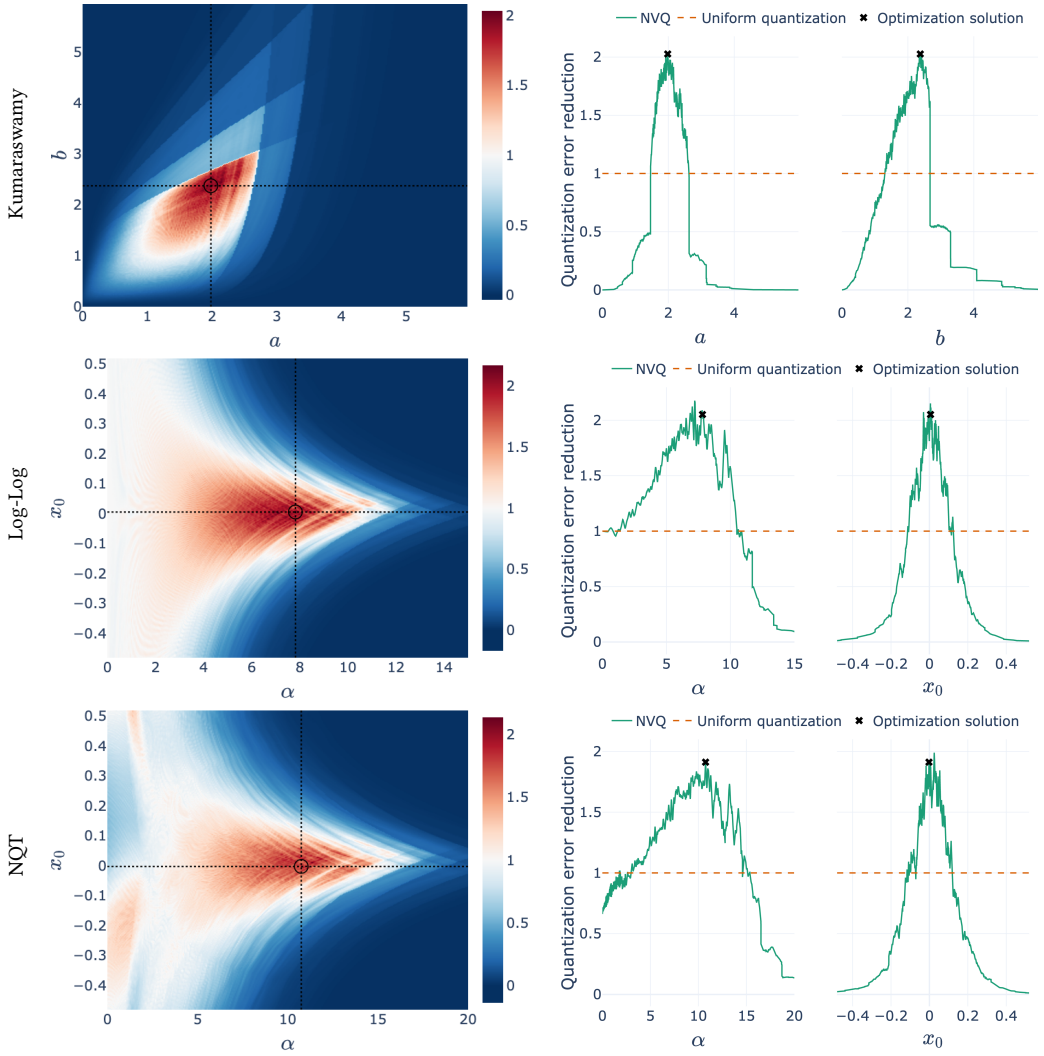

Fig. 13. Algorithm 1 finds a good maximum of Problem 6 using the nonlinearities presented in Section 4 for $\beta = 8$ bits. **Left:** the landscape of the objective function in Problem 6, which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), 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.

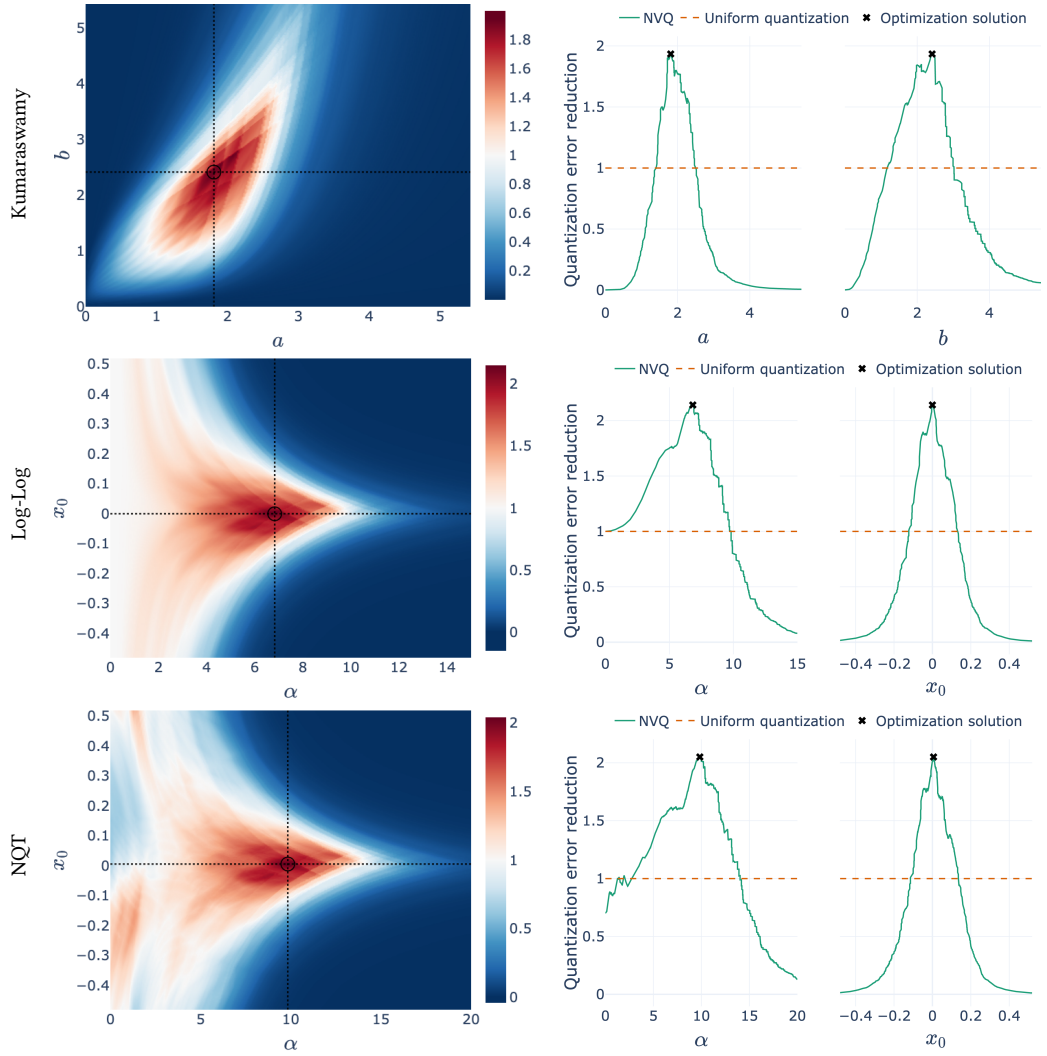

Fig. 14. Algorithm 1 finds a good maximum of Problem 6 using the nonlinearities presented in Section 4 for $\beta = 4$ bits. **Left:** the landscape of the objective function in Problem 6, which is the ratio of the MSE improvement over the uniform quantization (a value of 1 means parity, higher is better), 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.

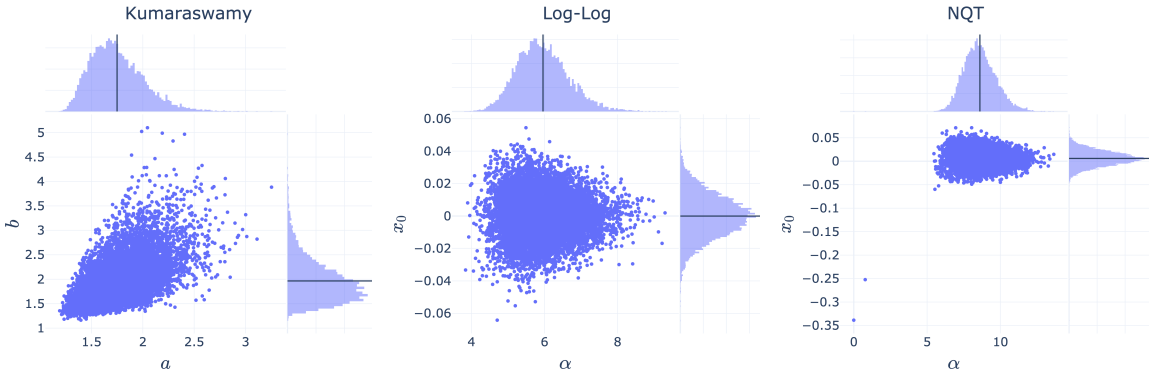

Fig. 15. The solutions for 10^4 vectors from gecko-100k. Different nonlinearity parameters are chosen for each vector.

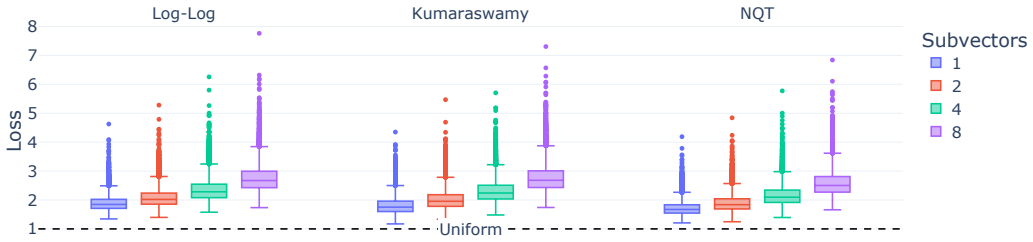

Fig. 16. Increasing the number of subvectors (depicted here for $\beta = 8$) increases the objective function (higher is better) in Problem 6 for 10^4 vectors from ada002-100k.

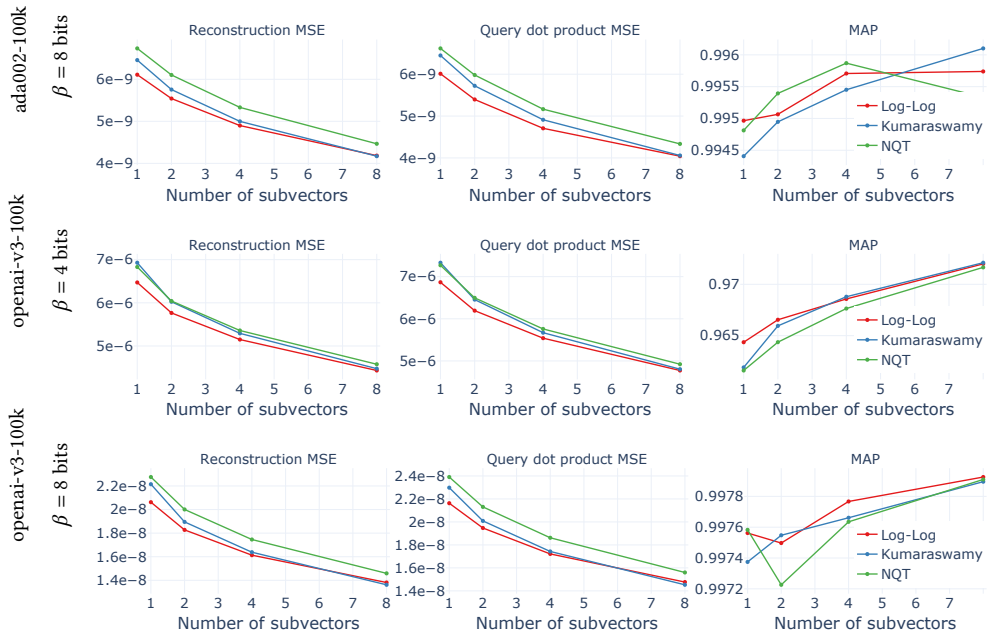

Fig. 17. Increasing the number of subvectors lowers the average reconstruction error and the query dot product error, defined as $\sum_{\boldsymbol{x}\in\mathcal{X}}(\langle \boldsymbol{q}, \boldsymbol{x}\rangle - \langle \boldsymbol{q}, \widetilde{\boldsymbol{x}}\rangle)^2$, and increases the mean average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