# Reproducibility

The backbone recommendation model, DLRM by Naumov et al. [2019], has an open-source PyTorch implementation available on Github which includes an implementation of CE. For CCE you need a fast library for K-means. We recommend the open-sourced implementation by Johnson et al. [2019] for better performance, but you can also use the implementation in Scikit-learn [Pedregosa et al., 2011]. The baseline result should be straightforward to reproduce as we closely follow the instructions provided by Naumov et al. [2019]. For the CE methods, we only need to change two functions in the code: `create_emb` and `apply_emb`. We suggest using a class for each CE method; see Figure 3. For the random hash function, one could use a universal hash function or `numpy.random.randint`.

**Measuring the Embedding Compression factor** The most important number from our experimental section is the "Embedding Compression" factor in Table 1. We measure this by training the model with different caps on of parameters in the embedding tables (See e.g. the x-axis in Figure 4a. E.g. if the Criteo Kaggle dataset has categorical features with vocabularies of sizes $10$, $100$ and $10^6$, we try e.g. to cap this at $8000$, using a full embedding table for the small features, and a CCE table with $8000/16 = 500$ rows (since each row has 16 parameters). This corresponds to a compression rate of $(10 + 100 + 10^6)/(10 + 100 + 500) \approx 1639.5$. Or if we measure only the compression of the largest table, $10^6/500 = 2000$. Unfortunately there's a discrepancy in the article, which we only found after the main deadline, that uses the second measure in the introduction (hence the number 11,000x compression) where as Figure 4a uses the first measure (and thus the lower number 8,5000x).

For the experiments where we only train for 1 epoch, some methods never reach baseline BCE within the number of parameters we test. Hence the Compression Rates we report are based on extrapolations. For each algorithm we report a range, e.g. 127-155x for CE with Concatenation on Criteo Kaggle 1 epoch. Since the loss graphs tend to be convex, the upper bound (155x) is based on a linear interpolation (being optimistic about when the method will hit baseline BCE) and the lower bound (127x) is based on a quadratic interpolation, which only intersects the baseline at a higher parameter count.

**K-means** For the K-means from FAISS, we use `max_points_per_centroid=256` and `niter=50`. The first parameter sub-samples the number of points to 256 times the number of cluster centroids ($k$), and is the recommended rate after which "no benefit is expected" according to the library maintainers. In practice we predict the right value will depend on the dimensionality of your data, so using the split into lower dimensional columns is beneficial. For niter we initially tried a larger value (300), but found it didn't improve the final test loss. We found on Kaggle for PQ, niter=50, BCE=$0.455540$ and niter=300, BCE=$0.455537$. For CCE (single epoch, single clustering at half an epoch), niter=50 gave BCE=$0.45928$ and niter=300 gave BCE=$0.45905$, so a very slight improvement, but not enough to make up for the extra training time.

**Datasets** For our experiments, we sub-sampled an eighth of the Terabyte dataset and pre-hashed them using a simple modulus to match the categorical feature limit of the Kaggle dataset. For both Kaggle and Terabyte dataset, we partitioned the data from the final day into validation and test sets. Using the benchmarking setting of the DLRM code, the Kaggle dataset has around 300,000 batches while the Terabyte dataset has around 2,000,000 batches.

**Early stopping** Early stopping is used when running the best of 10 epochs on the Kaggle dataset. We measure the performance of the model in BCE every 50,000 batches (around one-sixth of one epoch) using the validation set. If the minimum BCE of the previous epoch is less than the minimum BCE of the current epoch, we early stop.

**Deep Hash Embeddings** We follow Kang et al. [2021] in using a fixed-width MLP with Mish activation. However, DHE is only described in one version in the paper: 5 layers of 1024 nodes per layer. For our experiments, we need to initialize DHE with different parameter budgets. We found that in general, DHE performs better with fewer layers when the number of parameters is fixed. However, we cannot use just a single layer, since that would be a linear embedding table, not an MLP. As a compromise, we fix the number of hidden layers to 2 and set the number of hashes to be the same as the dimension of the hidden layers.

For example, if we were allowed to use $64,000$ parameters with an embedding dimension of 64, then by solving a quadratic equation we get that the number of hashes and the dimension of the hidden layers are both 136. This gives us

$$\text{n\_hashes} \cdot \text{hidden\_dim} + 2 * \text{hidden\_dim}^2 + \text{hidden\_dim} \cdot \text{embedding\_dim} = 64192$$

parameters.

# A What didn't work

Here are the ideas we tried but didn't work at the end.

**Using multiple helper tables** It is a natural idea use more than one helper table. However, in our experiments, the effect of having more helper tables is not apparent.

**Circular clustering** Based on the CE concat method, the circular clustering method would use information from other columns to do clustering. However, the resulting index pointer functions are too similar to each other, meaning that this method is essentially the hashing trick. We further discuss this issue in Appendix G.

**Continuous clustering** We originally envisioned our methods in a tight loop between training and (re)clustering. It turned out that reducing the number of clusterings didn't impact performance, so we eventually reduced it all the way down to just one. In practical applications, with distribution shift over time, doing more clusterings may still be useful, as we discuss in Section 3.

**Changing the number of columns** In general, increasing the number of columns leads to better results. However the marginal benefits quickly decrease, and as the number of hash functions grow, so does the training and inference time. We found that 4 columns / hash-functions was a good spot.

**Residual vector quantization** The CCE method combines Product Quantization (PQ) with the CE concat method. We tried combining Residual vector quantization (RVQ) with the Hash Embeddings method from Tito Svenstrup et al. [2017]. This method does not perform significantly better than the Hash Embeddings method.

**Seeding with PQ** We first train a full embedding table for one epoch, and then do Product Quantization (PQ) on the table to obtain the index pointer functions.

We then use the index pointer functions instead of random hash functions in the CE concat method. This method turned out performing badly: The training loss quickly diverges from the test loss after training on just a few batches of data.

Here are some variations of the CCE method:

**Earlier clustering** We currently have two versions of the CCE method: CCE half, where clustering happens at the middle of the first epoch, and CCE, where clustering happens at the end of the first epoch. We observe that when we cluster earlier, the result is slightly worse. Though in our case the CCE half method still outperforms the CE concat method.

**More parameters before clustering** The CCE method allows using two number of parameters, one in Step 1 where we follow the CE hybrid method to get a sketch, and one in Step 3 where we follow the CE concat method. We thought that by using more parameters at the beginning, we would be able to get a better set of index pointer tables. However, the experiment suggested that the training is faster but the terminal performance is not significantly better.

**Smarter initialization after clustering** In Algorithm 3 we initialize $M_i$ with the cluster centroids from K-means and the "helper table" $M_i' \leftarrow 0$. We could instead try to optimize $M_i'$ to match the residuals of $T$ as well as possible. This could reduce the discontinuity during training more than initializing to zeros. However, we didn't see a large effect in either training loss smoothness or the ultimate test score.

# B Proof of the main theorem

$$\begin{bmatrix} 0.417 & 0.720 \\ 0.000 & 0.302 \\ 0.147 & 0.092 \\ 0.186 & 0.346 \\ 0.397 & 0.539 \\ 0.419 & 0.685 \\ 0.204 & 0.878 \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.411 & 0.648 \\ 0.093 & 0.324 \\ 0.204 & 0.878 \\ 0.147 & 0.092 \end{bmatrix}$$

Figure 5: **K-means as matrix factorization.** A central part of the analysis of CCE is the simple observation that K-means factors a matrix into a tall sparse matrix and a small dense one. In other words, it finds a sparse approximation the column space of the matrix.

Let's remind ourselves of the "Dense CCE algorithm" from Section 3: Given $X \in \mathbb{R}^{n \times d_1}$ and $Y \in \mathbb{R}^{n \times d_2}$, pick $k$ such that $n > d_1 > k > d_2$. We want to solve find a matrix $T^*$ of size $d_1 \times d_2$ such that $\|XT^* - Y\|_F$ is minimized – the classical Least Squares problem. However, we want to use memory less than the typical $nd_1^2$. We thus use this algorithm:

**Dense CCE Algorithm:** Let $T_0 = 0 \in \mathbb{R}^{d_1 \times d_2}$. For $i = 1$ to $m$:

$$\begin{aligned} \text{Sample} \quad & G_i \sim N(0,1)^{d_1 \times (k-d_2)}; \\ \text{Compute} \quad & H_i = [T_{i-1} \mid G_i] \in \mathbb{R}^{d_1 \times k} \\ & M_i = \arg\inf_M \|XH_iM - Y\|_F^2 \in \mathbb{R}^{k \times d_2}. \\ & T_i = H_iM_i \end{aligned}$$

We will now argue that $T_m$ is a good approximation to $T^*$ in the sense that $\|XT_m - Y\|_F^2$ is not much bigger than $\|XT^* - Y\|_F^2$.

Let's consider a non-optimal choice of $M_i$ first. Suppose we set $M_i = \begin{bmatrix} I_{d_2} \\ M_i' \end{bmatrix}$ where $M_i'$ is chosen such that $\|H_iM_i - T^*\|_F$ is minimized. By direct multiplication, we have $H_iM_i = T_{i-1} + G_iM_i'$. Hence in this case minimizing $\|H_iM_i - T^*\|_F$ is equivalent to finding $M_i'$ at each step such that $\|G_iM_i' - (T^* - T_{i-1})\|_F$ is minimized.

In other words, we are trying to estimate $T^*$ with $\sum_i G_iM_i'$, where each $G_i$ is random and each $M_i'$ is greedily chosen at each step. This is similar to, for example, the approaches in Barron et al. [2008], though they use a concrete list of $G_i$'s. In their case, by the time we have $d_1/k$ such $G_i$'s, we are just multiplying $X$ with a $d_1 \times d_1$ random Gaussian matrix, which of course will have full rank, and so the concatenated $M$ matrix can basically ignore it. However, in our case we do a local, not global optimization over the $M_i$.

Recall the theorem:

**Theorem B.0.** *Given $X \in \mathbb{R}^{n \times d_1}$ and $Y \in \mathbb{R}^{n \times d_2}$. Let $T^* = \arg\min_{T \in \mathbb{R}^{d_1 \times d_2}} \|XT - Y\|_F^2$ be an optimal solution to the least squares problem. Then*

$$\mathrm{E}\left[\|XT_i - Y\|_F^2\right] \le (1-\rho)^{i(k-d_2)}\|XT^*\|_F^2 + \|XT^* - Y\|_F^2,$$

*where $\rho = \|X\|_{-2}^2 / \|X\|_F^2$.*

Here we use the notation that $\|X\|_{-2}$ is the smallest singular value of $X$.

**Corollary B.1.** *In the setting of the theorem, if all singular values of $X$ are equal, then*

$$\mathrm{E}\left[\|XT_i - Y\|_F^2\right] \le e^{-i\frac{k-d_2}{d_1}}\|XT^*\|_F^2 + \|XT^* - Y\|_F^2.$$

*Proof of Theorem B.1.* Note that $\|X\|_F^2$ is the sum of the $d_1$ singular values squared: $\|X\|_F^2 = \sum_i \sigma_i^2$. Since all singular values are equal, say to $\sigma \in \mathbb{R}$, then $\|X\|_F^2 = d_1\sigma^2$. Similarly in this setting, $\|X\|_{-2}^2 = \sigma^2$ so $\rho = 1/d_1$. Using the inequality $1 - 1/d_1 \le e^{-1/d_1}$ gives the corollary. □

*Proof of Theorem B.0.* First split $Y$ into the part that's in the column space of $X$ and the part that's not, $Z$. We have $Y = XT^* + Z$, where $T^* = \arg\min_T \|XT - Y\|_F$ is the solution to the least

squares problem. By Pythagoras theorem we then have

$$\mathrm{E}\left[\|XT_i - Y\|_F^2\right] = \mathrm{E}\left[\|XT_i - (XT^* + Z)\|_F^2\right] = \mathrm{E}\left[\|X(T_i - T^*)\|_F^2\right] + \|Z\|_F^2,$$

so it suffices to show

$$\mathrm{E}\left[\|X(T_i - T^*)\|_F^2\right] \leq (1-\rho)^{i(k-d_2)}\|XT^*\|_F^2.$$

We will prove the theorem by induction over $i$. In the case $i = 0$ we have $T_i = 0$, so $\mathrm{E}[\|X(T_0 - T^*)\|_F^2] = \mathrm{E}[\|XT^*\|_F^2]$ trivially. For $i \geq 1$ we insert $T_i = H_i M_i$ and optimize over $M_i'$:

$$\begin{aligned}
\mathrm{E}[\|X(T_i - T^*)\|_F^2] &= \mathrm{E}[\|X(H_i M_i - T^*)\|_F^2] \\
&\leq \mathrm{E}[\|X(H_i[I \mid M_i'] - T^*)\|_F^2] \\
&= \mathrm{E}[\|X((T_{i-1} + G_i M_i') - T^*)\|_F^2] \\
&= \mathrm{E}[\|X(G_i M_i' - (T^* - T_{i-1}))\|_F^2]. \\
&= \mathrm{E}[\mathrm{E}[\|X(G_i M_i' - (T^* - T_{i-1}))\|_F^2 \mid T_{i-1}]] \\
&\leq (1-\rho)^{k-d_2}\,\mathrm{E}[\|X(T^* - T_{i-1})\|_F^2] \\
&\leq (1-\rho)^{i(k-d_2)}\|XT^*\|_F^2,
\end{aligned}$$

where the last step followed by induction. The critical step here was bounding

$$\mathrm{E}_G[\inf_M \|X(GM - T)\|_F^2] \leq (1-\rho)^{k-d_2}\|XT\|_F^2,$$

for a fixed $T$. We will do this in a series of lemmas below. □

We show the lemma first in the "vector case", corresponding to $k = 2, d_2 = 1$. The general matrix case follow below, and is mostly a case of induction on the vector case.

**Lemma B.2.** *Let $X \in \mathbb{R}^{n\times d}$ be a matrix with singular values $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$. Define $\rho = \sigma_d^2 / \sum_i \sigma_i^2$, then for any $t \in \mathbb{R}^d$,*

$$\mathrm{E}_{g\sim N(0,1)^d}\left[\inf_{m\in\mathbb{R}} \|X(gm - t)\|_2^2\right] \leq (1-\rho)\|Xt\|_2^2.$$

*Proof.* Setting $m = \langle Xt, Xg\rangle / \|Xg\|_2^2$ we get

$$\begin{aligned}
\|X(gm - t)\|_2^2 &= m^2\|Xg\|_2^2 + \|Xt\|_2^2 - 2m\langle Xg, Xt\rangle &(1)\\
&= \left(1 - \frac{\langle Xt, Xg\rangle^2}{\|Xt\|_2^2\|Xg\|_2^2}\right)\|Xt\|_2^2. &(2)
\end{aligned}$$

We use the singular value decomposition of $X = U\Sigma V^T$. Since $g \sim N(0,1)^d$ and $V^T$ is unitary, we have $V^T g \sim N(0,1)^d$ and hence we can assume $V = I$. Then

$$\begin{aligned}
\frac{\langle Xt, Xg\rangle^2}{\|Xt\|_2^2\|Xg\|_2^2} &= \frac{(t^T\Sigma U^T U\Sigma g)^2}{\|U\Sigma t\|_2^2\|U\Sigma g\|_2^2} &(3)\\
&= \frac{(t^T\Sigma^2 g)^2}{\|\Sigma t\|_2^2\|\Sigma g\|_2^2} &(4)\\
&= \frac{(\sum_i t_i\sigma_i^2 g_i)^2}{(\sum_i \sigma_i^2 t_i^2)(\sum_i \sigma_i^2 g_i^2)}, &(5)
\end{aligned}$$

where Equation (4) follows from $U^T U = I$ in the SVD. We expand the upper sum to get

$$\begin{aligned}
\mathrm{E}_g\left[\frac{(\sum_i t_i\sigma_i^2 g_i)^2}{(\sum_i \sigma_i^2 t_i^2)(\sum_i \sigma_i^2 g_i^2)}\right] &= \mathrm{E}_g\left[\frac{\sum_{i,j} t_i t_j \sigma_i^2\sigma_j^2 g_i g_j}{(\sum_i \sigma_i^2 t_i^2)(\sum_i \sigma_i^2 g_i^2)}\right] &(6)\\
&= \mathrm{E}_g\left[\frac{\sum_i t_i^2\sigma_i^4 g_i^2}{(\sum_i \sigma_i^2 t_i^2)(\sum_i \sigma_i^2 g_i^2)}\right]. &(7)
\end{aligned}$$

Here we use the fact that the $g_i$'s are symmetric, so the cross terms of the sum have mean 0. By scaling, we can assume $\sum_i \sigma_i^2 t_i^2 = 1$ and define $p_i = \sigma_i^2 t_i^2$. Then the sum is just a convex combination:

$$(7) = \sum_i p_i \, \mathrm{E}_g \left[ \frac{\sigma_i^2 g_i^2}{\sum_i \sigma_i^2 g_i^2} \right]. \tag{8}$$

Since $\sigma_i \geq \sigma_d$ and $g_i$'s are IID, by direct comparison we have

$$\mathrm{E}_g \left[ \frac{\sigma_i^2 g_i^2}{\sum_i \sigma_i^2 g_i^2} \right] \geq \mathrm{E}_g \left[ \frac{\sigma_d^2 g_d^2}{\sum_i \sigma_i^2 g_i^2} \right]$$

Hence

$$(7) \geq \mathrm{E}_g \left[ \frac{\sigma_d^2 g_d^2}{\sum_i \sigma_i^2 g_i^2} \right] \sum_i p_i = \mathrm{E}_g \left[ \frac{\sigma_d^2 g_d^2}{\sum_i \sigma_i^2 g_i^2} \right].$$

It remains to bound

$$\mathrm{E}_g \left[ \frac{\sigma_d^2 g_d^2}{\sum_i \sigma_i^2 g_i^2} \right] \geq \frac{\sigma_d^2}{\sum_i \sigma_i^2} = \rho, \tag{9}$$

but this follows from a cute, but rather technical lemma, which we will postpone to the end of this section. (Theorem B.4.) □

It is interesting to notice how the improvement we make each step (that is $1 - \rho$) could be increased to $1 - 1/d$ by picking $G$ from a distribution other than IID normal.

If $X = U\Sigma V^T$, we can also take $g = V\Sigma^{-1}g'$, where $g' \sim N(0,1)^{d_1 \times (k-d_2)}$. In that case we get

$$\mathrm{E}\left( \frac{\langle Xt, Xg \rangle}{\|Xg\|_2 \|Xt\|_2^2} \right)^2 = \mathrm{E}\left( \frac{t^T V \Sigma^2 V^T g}{\|Ug'\|_2 \|Xt\|_2^2} \right)^2 = \mathrm{E}\left( \frac{t^T V \Sigma g'}{\|g'\|_2 \|Xt\|_2^2} \right)^2 = \frac{1}{d_1} \frac{\|t^T V \Sigma\|_2^2}{\|Xt\|_2^2} = \frac{1}{d_1}.$$

So this way we recreate the ideal bound from Theorem B.1. Note that $\frac{\|X\|_{-2}^2}{\|X\|_F^2} \leq 1/d_1$. Of course it comes with the negative side of having to compute the SVD of $X$. But since this is just a theoretical algorithm, it's still interesting and shows how we would ideally update $T_i$. See Figure 6 for the effect of this change experimentally.

It's an interesting problem how it might inspire a better CCE algorithm. Somehow we'd have to get information about the the SVD of $X$ into our sparse super-space approximations.

We now show how to extend the vector case to general matrices.

**Lemma B.3.** *Let $X \in \mathbb{R}^{n \times d_1}$ be a matrix with singular values $\sigma_1 \geq \cdots \geq \sigma_{d_1} \geq 0$. Define $\rho = \sigma_{d_1}^2 / \sum_i \sigma_i^2$, then for any $T \in \mathbb{R}^{n \times d_2}$,*

$$\mathrm{E}_{G \sim N(0,1)^{d_1 \times k}} \left[ \inf_{M \in \mathbb{R}^{k \times d_2}} \|X(GM - T)\|_F^2 \right] \leq (1 - \rho)^k \|XT\|_F^2.$$

*Proof.* The case $k = 1, d_2 = 1$ is proven above in Theorem B.2.

**Case $k = 1$:** We first consider the case where $k = 1$, but $d_2$ can be any positive integer (at most $k$). Let $T = [t_1 | t_2 | \ldots | t_{d_2}]$ be the columns of $T$ and $M = [m_1 | m_2 | \ldots | m_{d_2}]$ be the columns of $M$. Then the $i$th column of $X(GM - T)$ is $X(Gm_i - t_i)$, and since the squared Frobenius norm of a

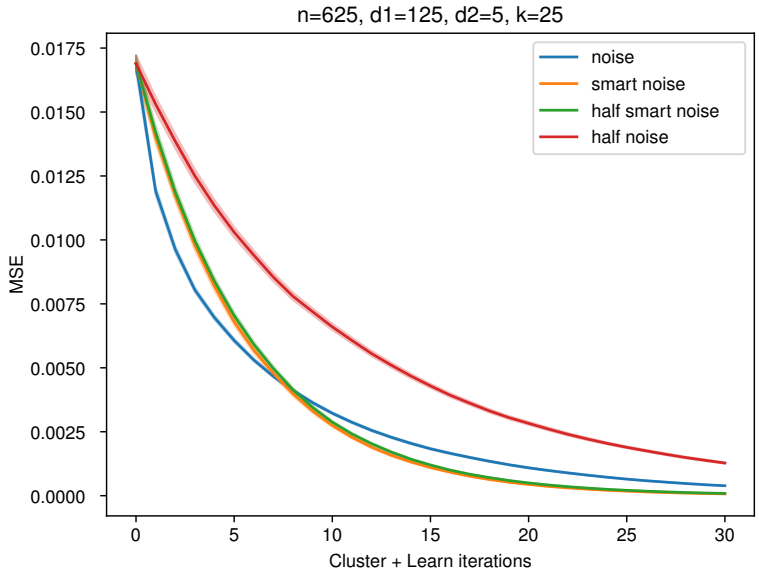



Figure 6: **SVD aligned noise converges faster.** In the discussion we mention that picking the random noise in $H_i$ as $g = V\Sigma^{-1}g'$, where $g' \sim N(0,1)^{d_1 \times (k-d_2)}$, can improve the convergence rate from $(1-\rho)^{ik}$ to $(1-1/d)^{ik}$, which is always better. In this graph we experimentally compare this approach (labeled "smart noise") against the IID gaussian noise (labeled "noise"), and find that the smart noise indeed converges faster – at least once we get close to zero noise. The graph is over 40 repetitions where $X$ is a random rank-10 matrix plus some low magnitude noise.
We also investigate how much we lose in the theorem by only considering $M$ on the form $[I|M']$, rather than a general $M$ that could take advantage of last rounds $T_i$. The plots labeled "half noise" and "half smart noise" are limited in this way, while the two others are not. We observe that the effect of this is much larger in the "non-smart" case, which indicates that the optimal noise distribution we found might accidentally be tailored to our analysis.

matrix is simply the sum of the squared column l2 norms, we have

$$\begin{aligned}
\mathrm{E}[\|X(GM-T)\|_F^2] &= \mathrm{E}\left[\sum_{i=1}^{d_2} \|X(Gm_i - t_i)\|_2^2\right] \\
&= \sum_{i=1}^{d_2} \mathrm{E}\left[\|X(Gm_i - t_i)\|_2^2\right] \\
&\leq \sum_{i=1}^{d_2} (1-\rho)\,\mathrm{E}[\|Xt_i\|_2^2] \qquad (10) \\
&= (1-\rho)\,\mathrm{E}\left[\sum_{i=1}^{d_2} \|Xt_i\|_2^2\right] \\
&= (1-\rho)\,\mathrm{E}[\|XT\|_F^2].
\end{aligned}$$

where in (10) we applied the single vector case.

**Case $k > 1$:** This time, let $g_1, g_2, \ldots, g_k$ be the columns of $G$ and let $m_1^T, m_2^T, \ldots, m_k^T$ be the *rows* of $M$.

We prove the lemma by induction over $k$. We already proved the base-case $k = 1$, so all we need is the induction step. We use the expansion of the matrix product $GM$ as a sum of outer products

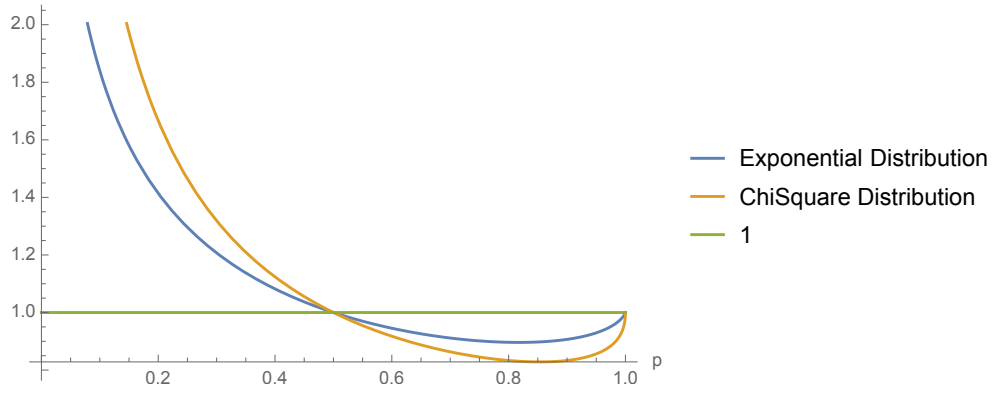



Figure 7: Expectation, $\mathrm{E}\left[\frac{x}{px+(1-p)y}\right]$ when $x, y$ are IID with Exponential (blue) or Chi Square distribution (Orange). In both cases the expectation is $\geq 1$ when $p \leq 1/2$, just as Theorem B.4 predicts.

$GM = \sum_{i=1}^{k} g_i m_i^T$:

$$\begin{aligned}
\mathrm{E}[\|X(GM - T)\|_F^2] &= \mathrm{E}\left[\left\|X\left(\sum_{i=1}^{k} g_i m_i^T - T\right)\right\|_F^2\right] \\
&= \mathrm{E}\left[\left\|X\left(g_1 m_1^T + \left(\sum_{i=2}^{k} g_i m_i^T - T\right)\right)\right\|_F^2\right] \\
&\leq (1-\rho)\,\mathrm{E}\left[X\left(\left\|\sum_{i=2}^{k} g_i m_i^T - T\right)\right\|_F^2\right] \qquad (11) \\
&\leq (1-\rho)^k\,\mathrm{E}\left[\|XT\|_F^2\right].
\end{aligned}$$

where (11) used the $k = 1$ case shown above, and (12) used the inductive hypothesis. This completes the proof for general $k$ and $d_2$ that we needed for the full theorem. □

### B.1 Technical lemmas

It remains to show an interesting lemma used for proving the vector case in Theorem B.2.

**Lemma B.4.** *Let $a_1 \ldots, a_n \geq 0$ be IID random variables and assume some values $p_i \geq 0$ st. $\sum_i p_i = 1$ and $p_n \leq 1/n$. Then*

$$E\left[\frac{a_n}{\sum_i p_i a_i}\right] \geq 1.$$

This completes the original proof with $p_i = \frac{\sigma_i^2}{\sum_j \sigma_j^2}$ and $a_i = g_i^2$.

*Proof.* Since the $a_i$ are IID, it doesn't matter if we permute them. In particular, if $\pi$ is a random permutation of $\{1, \ldots, n-1\}$,

$$E\left[\frac{a_n}{\sum_i p_i a_i}\right] = E_a\left[E_\pi\left[\frac{a_n}{p_n a_n + \sum_i p_i a_{\pi_i}}\right]\right] \tag{12}$$

$$\geq E_a\left[\frac{a_n}{E_\pi\left[p_n a_n + \sum_{i<n} p_i a_{\pi_i}\right]}\right] \tag{13}$$

$$= E_a\left[\frac{a_n}{p_n a_n + \sum_{i<n} p_i (\frac{1}{n-1}\sum_{j<n} a_j)}\right] \tag{14}$$

$$= E_a\left[\frac{a_n}{p_n a_n + (1-p_n)\sum_{i<n}\frac{a_i}{n-1}}\right], \tag{15}$$

where Equation (13) uses Jensen's inequality on the convex function $1/x$.

Now define $a = \sum_{i=1}^n a_i$. By permuting $a_n$ with the other variables, we get:

$$E_a\left[\frac{a_n}{p_n a_n + (1-p_n)\sum_{i<n}\frac{a_i}{n-1}}\right] = E_a\left[\frac{a_n}{p_n a_n + \frac{1-p_n}{n-1}(a-a_n)}\right] \tag{16}$$

$$= E_a\left[\frac{1}{n}\sum_{i=1}^n \frac{a_i}{p_n a_i + \frac{1-p_n}{n-1}(a-a_i)}\right] \tag{17}$$

$$= E_a\left[\frac{1}{n}\sum_{i=1}^n \frac{a_i/a}{\frac{1-p_n}{n-1} - (\frac{1-p_n}{n-1} - p_n)a_i/a}\right] \tag{18}$$

$$= E_a\left[\frac{1}{n}\sum_{i=1}^n \phi(a_i/a)\right], \tag{19}$$

where

$$\phi(q_i) = \frac{q_i}{\frac{1-p_n}{n-1} - (\frac{1-p_n}{n-1} - p_n)q_i}$$

is convex whenever $\frac{1-p_n}{n-1}/(\frac{1-p_n}{n-1} - p_n) = \frac{1-p}{1-np} > 1$, which is true when $0 \leq p_n < 1/n$. That means we can use Jensen's again:

$$\frac{1}{n}\sum_{i=1}^n \phi(a_i/a) \geq \phi\left(\frac{1}{n}\sum_i \frac{a_i}{a}\right) = \phi\left(\frac{1}{n}\right) = 1,$$

which is what we wanted to show. □

# C Hashing

If $h : [n] \to [m]$ and $s : [n] \to \{-1, 1\}$ are random functions, a Count Sketch is a matrix $H \in \{0, -1, 1\}^{m \times n}$ where $H_{i,j} = s(i)$ if $h(i) = j$ and 0 otherwise. Charikar et al. [2002] showed that if $m$ is large enough, the matrix $H$ is a dimensionality reduction in the sense that the norm $\|x\|_2$ of any vector in $\mathbb{R}^n$ is approximately preserved, $\|Hx\|_2 \approx \|x\|_2$.[4]

This gives a simple theoretical way to think about the algorithms above: The learned matrix $T' = H^T T$ is simply a lower dimensional approximation to the real table that we wanted to learn. While the theoretical result requires the random "sign function" $s$ for the approximation to be unbiased, in practice this appears to not be necessary when directly learning $T'$. Maybe because the vectors can simply be slightly shifted to debias the result.

There are many strong theoretical results on the properties of Count Sketches. For example, Woodruff [2014] showed that they are so called "subspace embeddings" which means the dimensionality

[4]This also implies that inner products are approximately preserved by the dimensionality reduction.

reduction is "robust" and doesn't have blind spots that SGD may accidentally walk into. However, the most practical result is that one only needs $h$ to be a "universal hash function" ala Carter and Wegman [1977], which can be as simple and fast as the "multiply shift" hash function by Dietzfelbinger et al. [1997].

If Count Sketch shows that hashing each $i \in [n]$ to a single row in $[m]$, we may wonder why methods like Hash Embeddings use multiple hash functions (or DHE uses more than a thousand.) The answer can be seen in the theoretical analysis of the "Johnson Lindenstrauss" transformation and in particular the "Sparse Johnson Lindenstrauss" as analyzed by Cohen et al. [2018]. The analysis shows that if the data being hashed is not very uniform, it is indeed better to use more than one hash function (more than 1 non-zero per column in $H$.) The exact amount depends on characteristics in the data distribution, but one can always get away with a sparsity of $\epsilon$ when looking for a $1 + \epsilon$ dimensionality reduction. Hence we speculate that DHE could in general replace the 1024 hash functions with something more like Hash Embeddings with an MLP on top. Another interesting part of the Cohen et al. [2018] analysis is that one should ideally split $[m]$ in segments, and have one hash function into each segment. This matches the implementations we based our work on below.

# D How to store the hash functions

We note that unlike the random hash functions used in Step 1, the index pointer functions obtained from clustering takes space *linear in the amount of training data* or at least in the ID universe size. At first this may seem like a major downside of our method, and while it isn't different from the index tables needed after Product Quantization, it definitely is something extra not needed by purely sketching based methods.

We give three reasons why storing this table is not an issue in practice:

1. The index pointer functions can be stored on the CPU rather than the GPU, since they are used as the first step of the model before the training/inference data has been moved from the CPU to the GPU. Furthermore the index lookup is typically done faster on CPUs, since it doesn't involve any dense matrix operations.
2. The index pointers can replace the original IDs. Unless we are working in a purely streaming setting, the training data has to be stored somewhere. If IDs are 64 bit integers, replacing them with four 16-bit index pointers is net neutral.
3. Some hashing and pruning can be used as a prepossessing step, reducing the universe size of the IDs and thus the size of the index table needed.

# E Different strategies for CCE

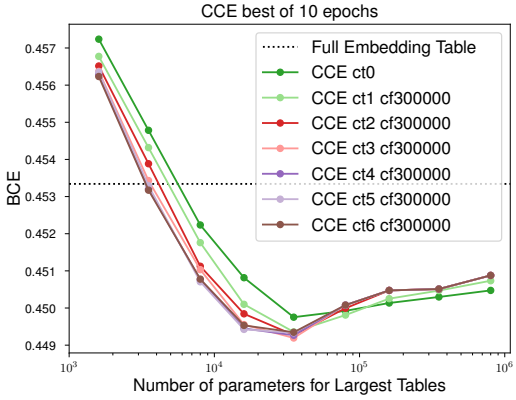
We include other graphs about CCE in Figure 8. They are all on the Kaggle dataset and were run three times. These graphs helped us develop insights on CCE and choose the correct versions for Figure 4a and Figure 4b.

# F AUC Graphs

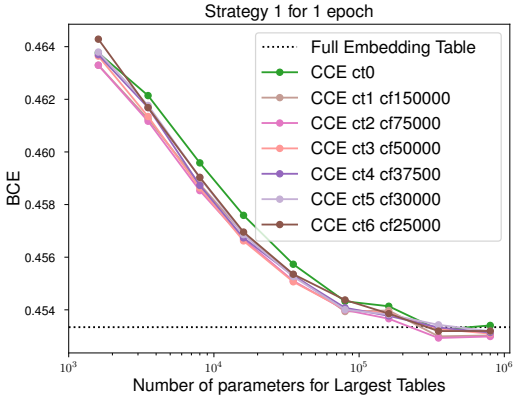
We also evaluate the models using AUC, which is another commonly used metric for gauging the effectiveness of a recommendation model. For example, it was used in [Kang et al., 2021]. AUC provides the probability of getting a correct prediction when evaluating a test sample from a balanced dataset. Therefore, a better model is implied by a larger AUC. In this section, we plot the graphs again using AUC; see Figure 9 and Figure 10.

# G Table Collapse

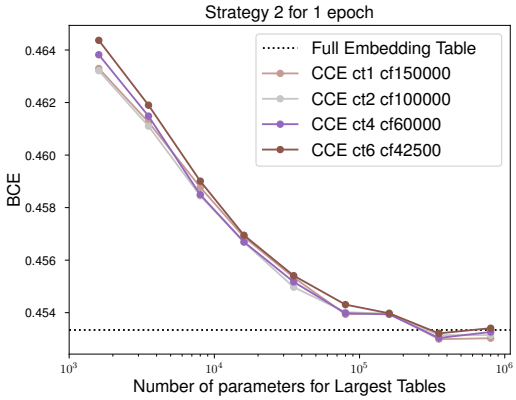
Table collapsing was a problem we encountered for the circular clustering method as described in Appendix A. We describe the problem and the metric we used to detect it here, since we think they may be of interest to the community.



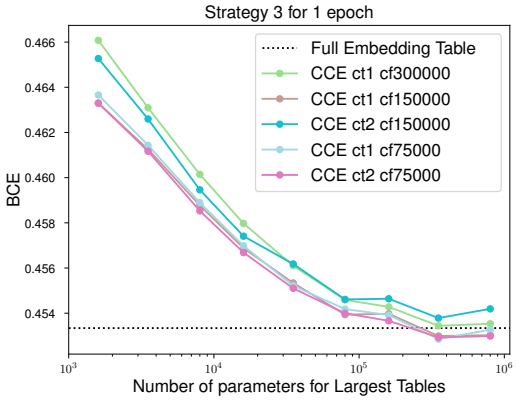

(a) **Kaggle dataset, CCE, best of 10 epoch**: We ran different versions of CCE for 10 epochs. Here ct means the number of clustering done, and cf refers to the number of batches between clusterings. Since each epoch has around $300,000$ batches, we essentially clustered once every epoch. The performance increases with more clustering. Another observation is that as $m$ increases, a few lines were merged due to early stopping. We found that CCE ct6 cf300000 performs the best, which becomes the CCE model in Figure 4a.




(b) **Kaggle dataset, CCE, Strategy 1**: We ran different versions of CCE for 1 epoch under the constraint that all clusterings must finish before half of an epoch. It turns out that there is a balance between the number of clusterings and the 'quality' of the clustering, represented by the number of batches between clusterings. We found that CCE ct2 cf75000 performs the best, which becomes the CCE model in Figure 4b.




(c) **Kaggle dataset, CCE, Strategy 2**: Strategy 2 here tries to have all the clustering finish by $2/3$ of an epoch. These runs did not perform well. It turned out that we need to let the model have time to converge after all the clusterings.



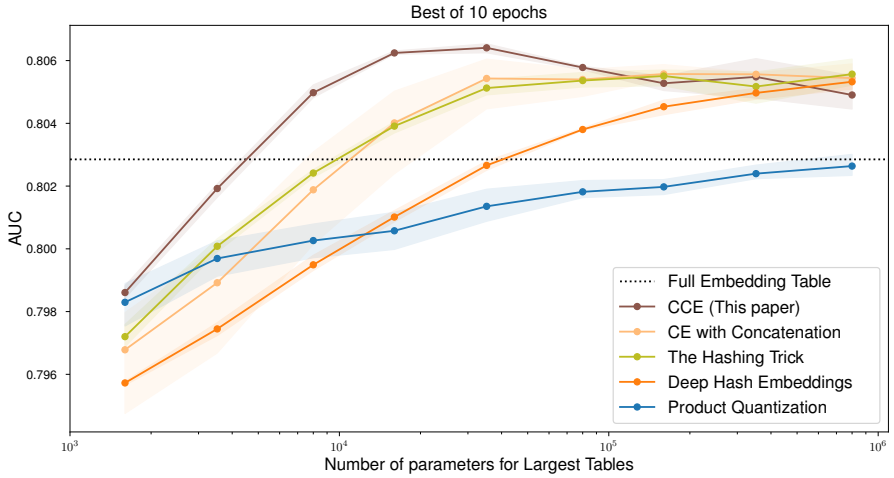

(d) **Kaggle dataset, CCE, Strategy 3**: This strategy perfectly summarizes all the previous findings. Increasing the number of clusterings in general gives better performance; Letting the model 'rest' after clustering increases the performance; Increasing the interval between clusterings give better result.

Figure 8: Strategies for CCE that gave us insight.

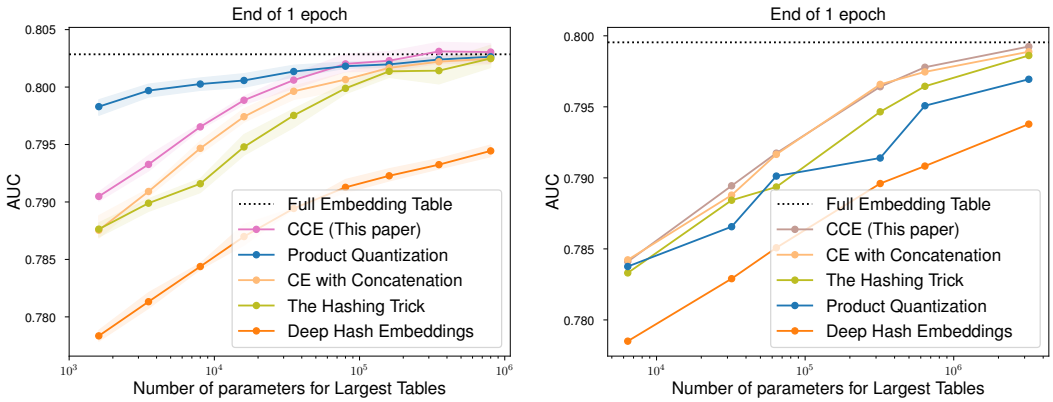
Suppose we are doing $k$-means clustering on a table of 3 partitions in order to obtain 3 index pointer functions $h_j^c$. These functions can be thought as a table, where the $(i,j)$-entry is given by $h_j^c(i)$.

There are multiple failure modes we have to be aware of. The first one is column-wise collapse:

$$\begin{array}{|c|c|c|}\hline 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 3 \\ \vdots & \vdots & \vdots \\ 1 & 3 & 1 \\ \hline \end{array}$$




(a) **Kaggle dataset, Best of 10 epoch, AUC**.




(b) **Kaggle dataset, 1 epoch, AUC**. (c) **Terabyte dataset, 1 epoch, AUC**.

Figure 9: The AUC version of Figure 4.

In this table the first column has collapsed to just one cluster. Because of the way $k$-means clustering works, this exact case of complete collapse isn't actually possible, but we might get arbitrarily low entropy as measured by $H_1$, which we define as follows: For each column $j$, its column entropy is defined to be the entropy of the probability distribution $p_j : h_j^c([n]) \to [0, 1]$ defined by

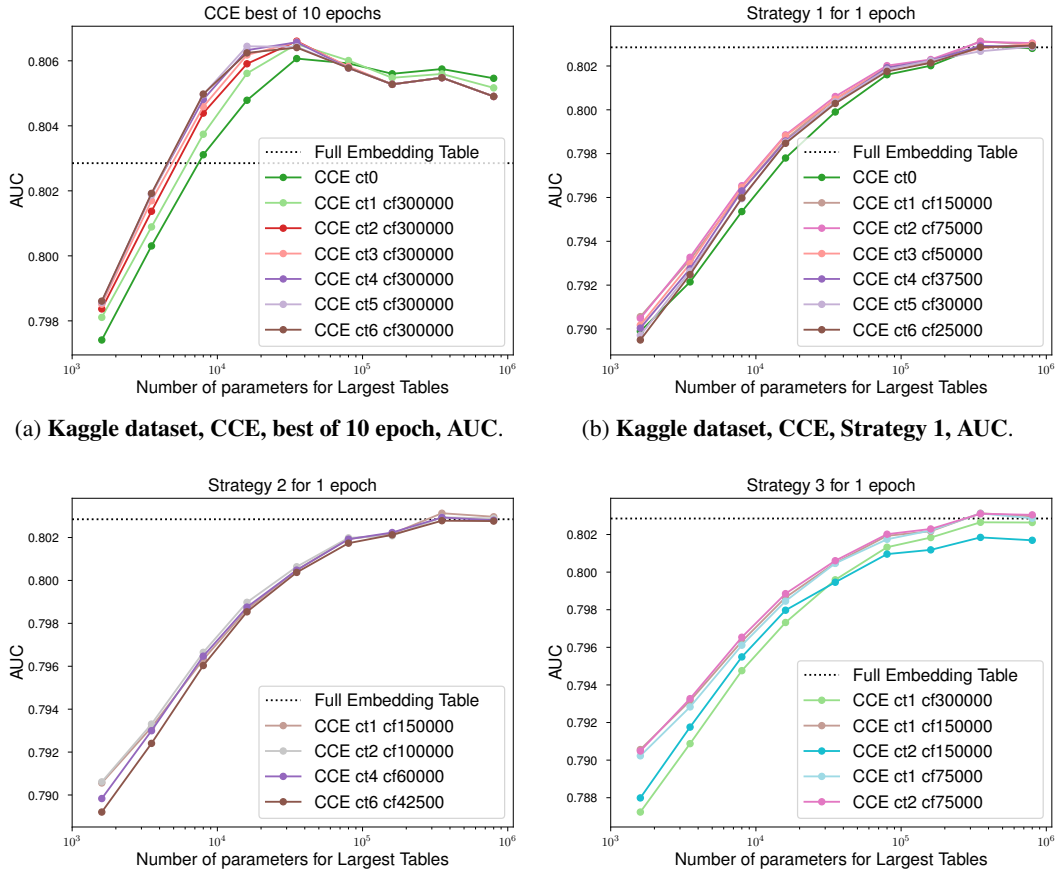
$$p_j(x) = \frac{\#\{i : h_j^c(i) = x\}}{n}.$$

Then we define $H_1$ to be the minimum entropy of the (here 3) column-entropies.

The second failure mode is pairwise collapse:

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 2 | 3 |
| 1 | 0 | 3 |
| 3 | 1 | 0 |
| 2 | 2 | 1 |

In this case the second column is just a permutation of the first column. This means the expanded set of possible vectors is much smaller than we would expect. We can measure pairwise collapse by computing the entropy of the histogram of pairs, where the entropy of the column pair $(j_1, j_2)$ is




(a) **Kaggle dataset, CCE, best of 10 epoch, AUC**.

(b) **Kaggle dataset, CCE, Strategy 1, AUC**.

(c) **Kaggle dataset, CCE, Strategy 2, AUC**.

(d) **Kaggle dataset, CCE, Strategy 3, AUC**.

Figure 10: The AUC version of Figure 8.

defined by the column entropy of $h^c_{j_1}(\cdot) + \max(h^c_{j_1})h^c_{j_2}(\cdot)$. Then we define $H_2$ to be the minimum of such pair-entropies for all $\binom{3}{2}$ pairs of columns.

Pairwise entropy can be trivially generalized to triple-wise and so on. If we have $c$ columns we may compute each of $H_1, \ldots, H_c$. In practice $H_1$ and $H_2$ may contain all the information we need.

### G.1 What entropies are expected?

The maximum value for $H_1$ is $\log k$, in the case of a uniform distribution over clusters. The maximum value for $H_2$ is $\log \binom{k}{2} \approx 2 \log k$. (Note $\log n$ is also an upper bound, where $n$ is the number of points in the dataset / rows in the table.)

With the CE method we expect all the entropies to be near their maximum. However, for the Circular Clustering method this is not the case! That would mean we haven't been able to extract any useful cluster information from the data.

Instead we expect entropies close to what one gets from performing Product Quantization (PQ) on a complete dataset. In short:

1. Too high entropy: We are just doing CE more slowly.
2. Too low entropy: We have a table collapse.
3. Golden midpoint: Whatever entropy normal PQ gets.