Appendix A DATASET DETAILS

As stated in Section 9.1, in our experiments, we consider 12 synthetic and real-world datasets commonly used as benchmarks for clustering tasks. The datasets we consider reflect diverse data distributions, clustering structures and application domains. Next, we describe each dataset in detail.

- The MNIST dataset [15] is a collection of 28×28 (vectorized) grayscale images of handwritten digits. We draw a stratified subsample of 25000 images, ensuring that the original class proportions across the ten digit categories are preserved. We rescale all pixel values by dividing by the maximum.
- The Double MNIST dataset is derived from MNIST by horizontally concatenating pairs of digit images. Each sample is a 28×56 (vectorized) grayscale image obtained by placing one 28×28 digit in the left position and another in the right position, and the label encodes the ordered pair of digits, yielding 100 classes in total. For our experiments, we generate 10000 such composite images using the procedure described above, resulting in an approximately uniform distribution over all digit pairs. Unlike the MNIST dataset, the Double MNIST dataset by construction admits a natural clustering with Khatri-Rao structure. We rescale all pixel values by dividing by the maximum.
- The HAR dataset, accessed through the UCI repository [12], consists of sensor readings collected from smartphone accelerometers and gyroscopes during human activity monitoring. Each sample is represented as a multivariate feature vector derived from raw time-series measurements, and the dataset includes 6 activity classes such as walking, standing and sitting. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The Olivetti Faces dataset, provided by scikit-learn [18], contains (vectorized) grayscale facial images of 40 individuals, with 10 images per subject captured under varying lighting conditions, facial expressions, and poses. Each image has resolution 64×64 pixels and is vectorized into a vector. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The Symbols dataset[4] consists of vectorized handwritten symbols. Each sample corresponds to a time series obtained from the drawing trajectory of a symbol. The symbols are drawn by 13 different individuals. The dataset contains a total of 1020 samples, each with 398 measurements. The symbols are organized into 6 natural clusters. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The stickfigures dataset [9] consists of synthetic (vectorized) silhouette images of human stick figures generated under varying poses. Examples of such images are given in Figure 1. Each image is provided at a fixed resolution and captures a simplified body configuration defined by joint positions and limb orientations, resulting in distinct pose-based classes. We rescale all pixel values by dividing by the maximum.
- The optdigits dataset, obtained via the UCI repository [12], contains 5620 (vectorized) handwritten digit images represented as 8×8 grayscale pixel grids. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The Classification dataset, sourced by scikit-learn [18], is a synthetic dataset. While it was originally designed for benchmarking classifiers, it is also useful for evaluating clustering algorithms which simply discard class labels until the evaluation of the clustering results. Each class corresponds to a cluster. Except in the experiments where we vary the number of data points, features, or clusters, the dataset consists of 5000 samples organized into 100 clusters, each corresponding to a single underlying cluster. The data are generated with 10 informative features, with no redundant or repeated features, ensuring that all dimensions contribute meaningfully to class separability. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The Chameleon dataset, sourced from the clustbench Benchmark Suite [8], consists of 10000 two-dimensional point clouds exhibiting complex, nonconvex cluster shapes with varying densities [11]. Essentially, the dataset corresponds to the configuration shown at the bottom of Figure 4, augmented with a substantial proportion of uniformly distributed noise points that do not belong to any natural cluster.
- The Soybean Large dataset, obtained via the UCI repository [12], is a dataset of categorical features. It consists of 562 plant samples belonging to one of 15 classes. For each plant, there are 35 observed categorical attributes describing the plant. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The Blobs dataset, sourced by scikit-learn [18], is a synthetic dataset that consists of samples grouped around a specified number of isotropic Gaussian clusters, specifically designed for controlled evaluation of clustering algorithms. Each cluster has standard deviation 1. The dataset consists of 5000 2-dimensional data points, arranged in 100 clusters, except in the experiments where we vary the number of data points, features of clusters. We standardize each feature by subtracting its mean and dividing by its standard deviation.
- The R15 dataset, accessed through the clustbench Benchmark Suite [8], consists of 600 two-dimensional points forming 15 well-separated Gaussian clusters. The cluster centroids in R15 are not arranged on a regular grid and therefore exhibit non-uniform inter-cluster distances. We standardize each feature by subtracting its mean and dividing by its standard deviation.

[4]The dataset is available at: https://www.timeseriesclassification.com/index.php

Appendix B IMPLEMENTATION DETAILS

In this section, we briefly discuss implementation details.

Implementation details of the naïve approach to Khatri-Rao clustering. For the naïve Khatri-Rao clustering baseline used in our experiments, we first run the SCIKIT-LEARN [18] implementation of standard k-MEANS to extract the desired number of cluster centroids. For instance, if the goal is to extract two sets of h_1 and h_2 protocentroids, k-MEANS retrieves $h_1 h_2$ clusters. Then, using a coordinate-descent procedure implemented in Python with closed-form updates, we decompose a set of centroids into two smaller sets of protocentroids whose Khatri–Rao product approximates the original centroids. The coordinate-descent procedure alternates between updating the protocentroids of the first and second set, holding the other set fixed. At each step, we use the closed-form updates illustrated in Section 5. The procedure stops either when a maximum number of iteration is reached (5000 by default) or when the total sum of squared difference between the initial centroids and the Khatri-Rao product of the protocentroids becomes smaller than a user-specified threshold (10^{-4} by default).

Upon termination of the described coordinate-descent procedure, we have the output sets of protocentroids and the corresponding centroids are readily obtained by aggregating protocentroids. To conclude, we assign each data point to the closest centroid. The implementation is available online in our code repository [5].

Implementation details of standard k-MEANS. We rely on the well-established SCIKIT-LEARN implementation of standard k-MEANS. The two k-MEANS baselines (namely k-MEANS with $h_1 + h_2$ centroids and $h_1 h_2$ centroids) are obtained by simply specifying the number of centroids $h_1 + h_2$ and $h_1 h_2$ as input. In the scalability experiments, instead, to ensure a fair comparison, we use an implementation of k-MEANS which mirrors the implementation of KHATRI-RAO-k-MEANS described next.

Implementation details of KHATRI-RAO-k-MEANS. Our experiments rely on a simple Python implementation of KHATRI-RAO-k-MEANS that is purely built on NUMPY [6], taking advantage of vectorized operations for efficiency. Such implementation is available online [7].

For initialization, by default we sample random data points as the initial protocentroids (as in Algorithm 1). Alternatively, we can adopt the strategy inspired from k-MEANS++ described in Section 6, which selects representative data points based on squared-distance criteria. Our implementation of this strategy either deterministically chooses the data point farthest from the previously selected centroids, or samples data points with probability proportional to their distance from those centroids.

After initialization, we iteratively compute assignments, protocentroid and corresponding centroids. Thanks to the closed-form updates introduced in Section 6, the updates of protocentroid (and centroids) are implemented in a fully-vectorized manner.

At each iteration we monitor convergence by tracking the movement of all reconstructed centroids; the algorithm terminates once this movement falls below a user-specified threshold or a maximum number of iteration is reached.

During the execution of the algorithm, in case empty clusters arise, they are handled by reinitializing the corresponding protocentroid to a random data point.

[5]https://github.com/maciap/KhatriRaoClustering/blob/main/scripts/KRkmeansExperimentsLib.py

[6]https://numpy.org/

[7]https://github.com/maciap/KhatriRaoClustering/tree/main/KathriRaokMeans

Our implementation is deliberately simple and easy to follow. In the future, more optimized or parallelized implementations could be developed for larger-scale settings.

KHATRI-RAO-k-MEANS admits both a time-efficient and a memory-efficient implementation. Algorithm 1 presents the memory-efficient implementation that avoids storing the full set of centroids by computing them on the fly from the stored set of protocentroids. This approach can reduce memory requirements, particularly when the number of clusters is large since memory requirements only grow additively with the total number of protocentroids instead of multiplicatively. For example, $h_1 h_2$ clusters demand storing only $h_1 + h_2$ protocentroids instead of $h_1 h_2$ centroids.

However, computing centroids on the fly incurs a runtime overhead. A time-efficient implementation is obtained by computing centroids once and storing them.

Implementation details of standard deep clustering algorithms. For DKM and IDEC, we rely on the off-the-shelf Pytorch-based implementations provided by by the ClustPy library [8].

Implementation details of Khatri-Rao deep clustering algorithms. For the implementation of Khatri-Rao deep clustering algorithms, we build the implementation of KHATRI-RAO DKM and KHATRI-RAO IDEC on top of the ClustPy implementations of the corresponding standard deep clustering algorithms.

Extending the ClustPy implementation of a standard deep clustering algorithm to the Khatri-Rao clustering paradigm is straightforward. During initialization, we rely on our implementation of KHATRI-RAO-k-MEANS to obtain initial protocentroids. We then reparameterize the centroids to satisfy the Khatri-Rao structure and adjust the autoencoder parameters to conform to the Hadamard-decomposition reparameterization. Our implementation of Khatri-Rao deep clustering algorithms is available online [9].

Appendix C ADDITIONAL EXPERIMENTS

This section reports additional experimental results that complement the evaluation presented in Section 9. The goal of these supplementary analyses is to provide a more comprehensive empirical picture of the behavior of Khatri-Rao clustering. First, we empirically assess the scalability of KHATRI-RAO-k-MEANS by measuring runtime and peak memory usage as the clustering tasks grow in size. Second, we study the effect of the number of centroids on the performance of KHATRI-RAO-k-MEANS and the corresponding baselines. Finally, we turn to deep clustering and evaluate how compressing the autoencoder via the Hadamard decomposition affects performance and training stability. Unless specified otherwise, all experimental settings are the same as for the experiments presented in Section 9.

Scalability. The goal of Khatri-Rao clustering is to extract accurate data summaries that are more succinct than those extracted by standard-clustering methods. Our experiments show that Khatri-Rao clustering can yield more succinct clustering-based summaries while maintaining comparable accuracy. However, it is also important to ensure that the advantages provided by Khatri-Rao clustering are not undermined by significantly worse scalability. Section 6 discusses the theoretical asymptotic time and space complexity of

[8]https://clustpy.readthedocs.io/en/v0.0.2/

[9]https://github.com/maciap/KhatriRaoClustering/tree/main/KhatriRaoDeepClustering

Khatri-Rao-k-Means. According to the discussion, Khatri-Rao-k-Means is expected to incur a runtime overhead over k-Means. However, Khatri-Rao-k-Means has the same asymptotic time complexity as k-Means. In addition, Khatri-Rao-k-Means can reduce memory usage of k-Means when the number of clusters and hence centroids to be represented grows. Our empirical analysis confirms the expectations derived from the time and space complexity discussion presented in Section 6.

The results of the empirical scalability analysis are summarized in Figure 10 and 11. The figures show runtime (in seconds) and peak memory usage (in Mebibytes) as the number of data points, features and centroids increase for the Blobs and Classification datasets. In particular, we vary the number of data points in $\{2500, 5000, 10000, 15000\}$ with 100 clusters and 100 features, we vary the number of features in $\{10000, 15000, 20000, 25000\}$ with 1000 data points and 100 clusters and we vary the number of clusters in $\{2500, 5000, 7500, 10000\}$ with 20000 data points and 100 features.

The results suggest that Khatri-Rao clustering introduces a runtime overhead compared to standard clustering, but this overhead does not significantly increase as the problem size increases. Instead, the overhead remains nearly constant as the number of data points, features and centroids increase.

The two-phase naïve approach to Khatri-Rao k-Means is usually the slowest algorithm, except when the number of centroids becomes large and Khatri-Rao-k-Means can become slower than the naïve approach to Khatri-Rao k-Means. On the other hand, standard k-Means is consistently the fastest algorithm.

As for memory requirements, Khatri-Rao-k-Means often incurs similar memory requirements as standard k-Means that use $h_1 + h_2$ vectors to represent centroids. On the other hand, kmeans with $h_1 h_2$ can be more memory demanding, particularly when the number of centroids grows. Specifically, as the number of centroids increases, k-Means with $h_1 h_2$ centroids uses up to about 1.6 and 1.8 times as much memory as Khatri-Rao-k-Means in the Blobs and Classification datasets, respectively. Furthermore, the gap grows with the number of centroids, corroborating the space-complexity discussion in Section 6.

It is also of interest to investigate the scalability of Khatri-Rao-k-Means as a function of the number of protocentroids sets. For the same experimental settings as in Figure 7, Figure 12 reports the runtime and memory usage of Khatri-Rao-k-Means as the number p of protocentroids sets varies in 2, 3, 4, while keeping a total of 12 vectors to represent the centroids. As suggested by the results, increasing the number of protocentroids leads to higher runtime, as more centroids are computed. Memory usage may also increase slightly, with variations always bounded by 5 mebibytes. For reference, we additionally report the runtime and memory usage of all baseline methods with $h_1 = h_2$. These measurements are obtained on the original Blobs and Classification datasets, corresponding to relatively small problem instances.

Impact of the number of centroids. Figure 13 shows the inertia incurred by the Khatri-Rao-k-Means and the baselines under comparison as the number of ground-truth clusters k in the data is varied within $\{25, 100, 225, 400, 625\}$ in the Blobs and Classification datasets. The figure demonstrates that Khatri-Rao-k-Means consistently achieves significantly lower inertia than the baselines at parity parameters (i.e., number of vectors to represent centroids). Standard k-Means with $h_1 h_2$ vectors to represent centroids achieves the lowest inertia, although its advantage over Khatri-Rao-k-Means, which uses only $h_1 + h_2$ vectors, remains consistently small.

Finally, the naïve approach always incurs the highest inertia, and its performance tend to worsen with the number of clusters, which is not the case for the other algorithms under comparison.

Impact of the Hadamard reparameterization on the autoencoder. As explained in Section 4.3, in our work, we reparameterize the weights of the autoencoder used in Khatri-Rao deep clustering as Hadamard product of low-rank factors. This reparameterization is also referred to as Hadamard reparameterization.

However, imposing the Hadamard reparameterization to reduce parameters and compute always introduces a bottleneck that could bias the model towards simpler solutions. Furthermore, excessive compression via Hadamard reparameterization can create challenges in training (optimization) and, in general, deteriorate clustering performance [13]. Therefore, it is important to carefully design the details of the Hadamard reparameterization such as the number of factors and the rank of each factor.

To ensure stable optimization, we use only two factors and we ensure that the rank of both factors is large enough so that the compressed autoencoder incurs reconstruction error similar to the uncompressed autoencoder. In our experiments, we also do not apply the Hadamard reparameterization to the input and output layers in the autoencoder, but only to the internal layers.

It is also important to evaluate the impact of the reparameterization on training stability and on the clustering performance. Thus, in the following, we empirically investigate the clustering performance and training stability as a function of the rank of both Hadamard factors. In particular, the rank of the Hadamard factors is varied in $\{20, 30, 40, 50, 60\}$.

Figure 14 displays the value of the clustering quality metrics monitored in this work for the Blobs and Classification datasets against the rank of each of two Hadamard factors. As the figure suggests, the chosen rank can have an impact on the clustering performance of Khatri-Rao deep clustering. However, such impact is limited. Increasing the rank of both Hadamard factors from 10 to 60 results in changes of at most 0.1 across all metrics, with most variations remaining below 0.05. On the Blobs dataset, the smallest rank considered (20) yields better clustering performance than higher ranks. In this case, the implicit regularization induced by the Hadamard reparameterization may be beneficial for clustering.

For the Blobs and Classification datasets, Figure 15 depicts the evolution of the training loss over 500 epochs for both the uncompressed autoencoder and the compressed autoencoder with Hadamard reparameterization in case the rank of both Hadamard factors is 20, 40 and 60. Although, particularly for the lower ranks, the compressed autoencoder exhibits a slightly slower initial decrease in training loss compared to the uncompressed autoencoder, it quickly converges to a comparable loss level. Overall, the training loss decays smoothly and consistently over iterations for both the uncompressed and compressed autoencoders across all considered ranks.

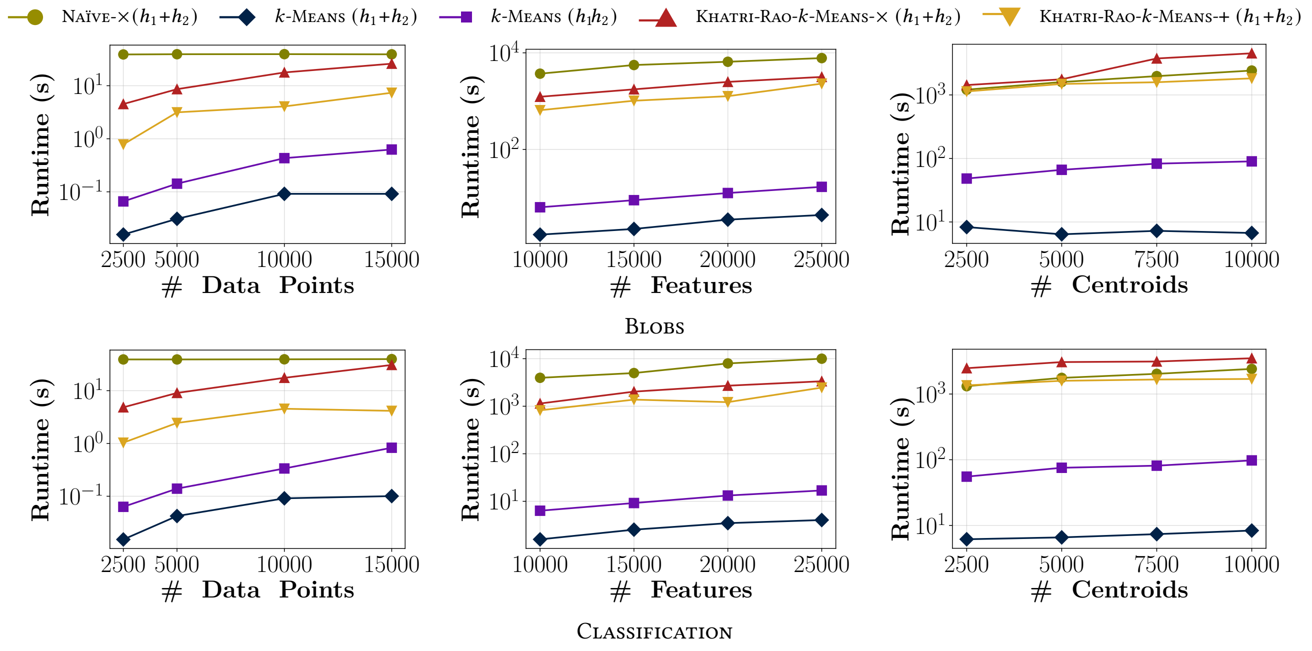

Figure 10: Experiments on Blobs and Classification datasets. Runtime (in seconds) by number of data points n, number of features m and number of centroids k in the data. Here, $h_1 = h_2 = \text{round}(\sqrt{k})$. The y-axis is on log scale.

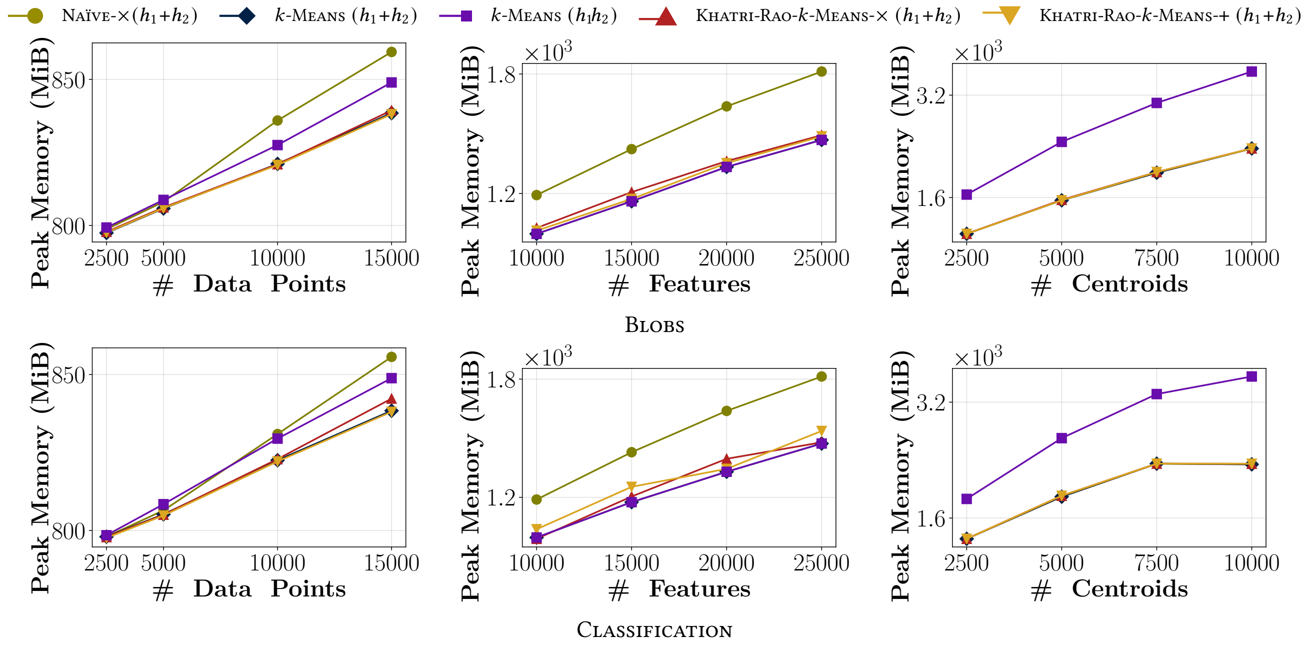

Figure 11: Experiments on Blobs and Classification datasets. Peak memory usage (in Mebibytes) by number of data points n, number of features m and number of centroids k in the data. Here, $h_1 = h_2 = \text{round}(\sqrt{k})$. The y-axis is on log scale.

Appendix D KHATRI-RAO CLUSTERING AND MATRIX DECOMPOSITION

As anticipated in Section 2, clustering is closely related to matrix decomposition. In general, both clustering and matrix decomposition pursue the objective of summarizing data through a succinct

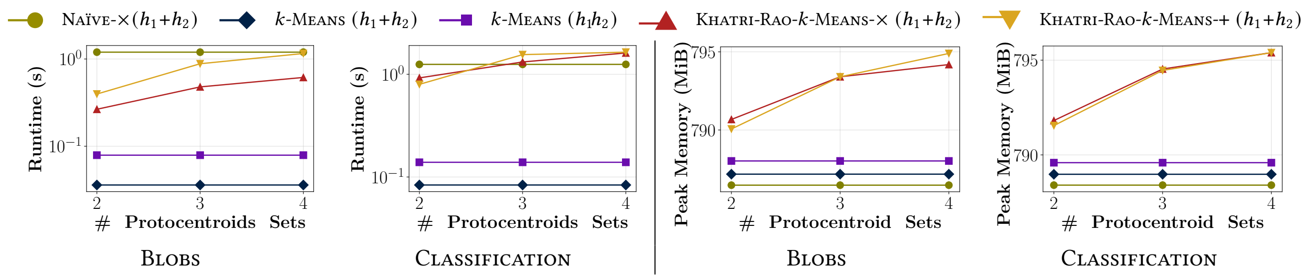

Figure 12: Experiments on Blobs and Classification datasets. Runtime in seconds (left) and memory usage in Mebibytes (right) by number of sets of protocentroids. All baselines assume $h_1 = h_2 = 6$.

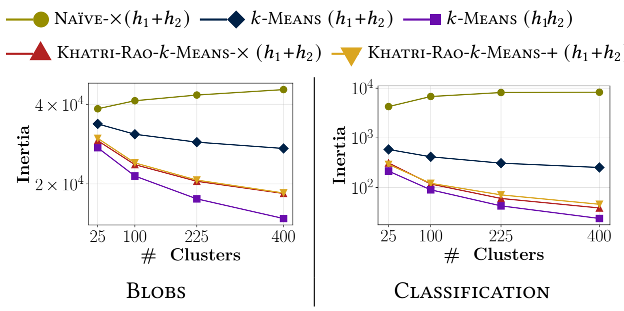

Figure 13: Experiments on Blobs and Classification datasets. Inertia as a function of the number of ground-truth clusters k for different algorithms using the same number of centroid parameters except for k-Means with h_1h_2 centroids which uses more parameters. Here, $h_1 = h_2 = \sqrt{k}$. The y-axis is on a logarithmic scale.

set of prototypes. Many clustering problems like k-Means can be seen as a particular case of matrix decomposition where one of the factor matrices is constrained to indicate cluster membership. In fact, it has been shown that the continuous relaxation of the k-Means cluster-membership indicators is given by the principal components of the data, establishing a direct link between k-Means clustering and principal component analysis, a prominent matrix-decomposition technique [7]. Clustering via matrix decomposition has been successfully applied in applications in different domains like text mining [20], graph mining [1, 13] and bioinformatics [2].

The established relationship between clustering and matrix factorization also extends to tensor data. Tensor decompositions can be used for clustering tensor data [14], with successful applications in areas like text mining across multiple languages [4], multilayer graph mining [3] and bioinformatics [19].

Given the close interplay between matrix (or tensor) decomposition and clustering, a natural question is how Khatri-Rao clustering fits within the broader framework of matrix decomposition.

In this section, we first review how standard k-Means clustering can be formulated as a matrix-decomposition problem. Second, we show that similarly also Khatri-Rao k-Means can be interpreted as a constrained instance of matrix decomposition. Then, we discuss the prominent example of nonnegative matrix factorization [17].

Finally, we present a simple illustrative experiment which anticipates the potential advantages of Khatri-Rao clustering in matrix decomposition–based clustering.

k-Means clustering as matrix factorization. The k-Means clustering problem can be formulated as approximating a data matrix by a product of lower-rank factors: one factor encodes cluster centroids and the other encodes cluster-membership assignments.

More specifically, the k-Means problem admits a natural reformulation as a constrained matrix decomposition task as follows. Let $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^m$ be the dataset and denote by $\Theta = \{\boldsymbol{\mu}_i\}_{i=1}^k$ the set of centroids, where $\boldsymbol{\mu}_i$ is the centroid of C_i. Introduce the (hard) assignment matrix $\mathbf{Z} \in \{0,1\}^{n\times k}$, where $\mathbf{Z}_{ji} = 1$ if $\mathbf{x}_j \in C_i$ and 0 otherwise, so that each data point is assigned to exactly one cluster (i.e., $\mathbf{Z1}_k = \mathbf{1}_n$.). Let $\mathbf{C} \in \mathbb{R}^{k\times m}$ be the centroid matrix whose i-th row equals $\boldsymbol{\mu}_i^\top$. Collecting the data points in $\mathcal{D}$ as the rows of $\mathbf{D} \in \mathbb{R}^{n\times m}$, the inertia objective in Equation (1), minimized in k-Means, can be written compactly as:

$$Q_C(\mathcal{D}, \Theta) = \|\mathbf{D} - \mathbf{ZC}\|_F^2, \tag{8}$$

under the assignment constraints encoded in $\mathbf{Z}$. Thus, k-Means problem is equivalent to seeking a decomposition $\mathbf{D} \approx \mathbf{ZC}$ where $\mathbf{Z}$ is required to be a binary assignment matrix with one non-zero entry per row.

Khatri-Rao k-Means clustering as constrained matrix decomposition. Like standard k-Means clustering, also Khatri-Rao k-Means clustering admits a formulation in terms of matrix decomposition.

To derive the formulation of Khatri-Rao k-Means as matrix decomposition, let us introduce matrices $\mathbf{Z}_i$ and $\mathbf{P}_i$ with $i = 1, \dots p$. Similarly to $\mathbf{Z}$ for standard k-Means, matrices $\mathbf{Z}_i$ have one-hot assignment vectors as rows. Matrix $\mathbf{Z}_i$ assigns each data point to one of h_i protocentroids stored as rows of matrix $\mathbf{P}_i$.

Then, the Khatri-Rao k-Means problem amounts to the minimization of:

$$Q_C(\mathcal{D}, \Theta) = \|\mathbf{D} - (\mathbf{Z}_1\mathbf{C}_1) \oplus (\mathbf{Z}_2\mathbf{C}_2) \dots \oplus (\mathbf{Z}_p\mathbf{C}_p)\|_F^2. \tag{9}$$

Hence, Khatri-Rao k-Means clustering induces a decomposition of the form $\mathbf{D} \approx (\mathbf{Z}_1\mathbf{C}_1) \oplus (\mathbf{Z}_2\mathbf{C}_2) \dots \oplus (\mathbf{Z}_p\mathbf{C}_p)$. Such decomposition is illustrated in Figure 16.

Equation 9 highlights the connection between Khatri-Rao k-Means and the Hadamard decomposition, studied in our previous work [6].

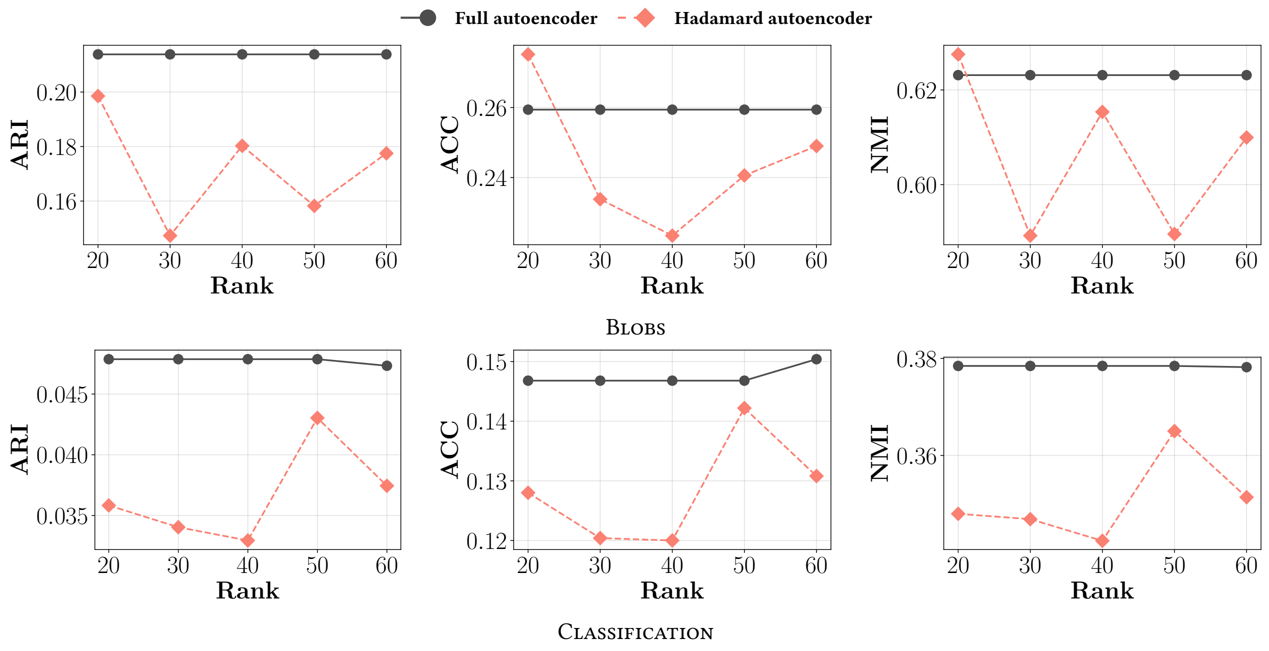

Figure 14: Experiments on Blobs and Classification datasets. Unsupervised clustering accuracy (ACC), adjusted Rand index (ARI) and normalized mutual information (NMI) for DKM and its Khatri-Rao extension against rank of both factors in the reparameterization of the autoencoder weights as Hadamard product of two low-rank factors.

In the Hadamard decomposition, and, in general, in matrix decomposition, the factors $\mathbf{Z}_i$ are unconstrained. As a result, summation does not introduce additional structure: the sum of q rank-r factors can represent matrices of rank at most qr, and summing two decompositions each using h vectors per factor is, in principle, equivalent to a single decomposition with $2h$ vectors per factor. In the setting of matrix decomposition, summation merely reallocates expressiveness across factors. By contrast, in matrix decomposition, the Hadamard product can yield a more expressive model, as the product of q rank-r factors can represent matrices of rank up to r^q, which explains the improved compression rates observed for the Hadamard decomposition in previous work [6, 10]. This expressiveness advantage is also the reason why we adopt the Hadamard reparameterization for Khatri-Rao deep clustering.

The expressiveness advantage of the product operator over the sum in matrix decomposition, however, does not carry over to Khatri-Rao clustering. Once the factors $\mathbf{Z}_i$ are constrained to be a one-hot cluster indicator, multiple decompositions can no longer be absorbed into a single enlarged factor. The discrete assignment structure prevents expressiveness from being freely reallocated, so that summation can introduce genuinely new modeling capacity.

The example of nonnegative matrix factorization. Nonnegative matrix factorization (NMF) exemplifies how clustering and matrix decomposition are fundamentally intertwined.

NMF represents data as a product of nonnegative factors [16], imposing the model $\mathbf{D} = \mathbf{W}\mathbf{H}$ with $\mathbf{W}_{i,j} \geq 0\ \forall i = 1 \ldots n, j = 1 \ldots m$ and $\mathbf{H}_{i,j} \geq 0\ \forall i = 1 \ldots n, j = 1 \ldots m$. NMF has emerged as a powerful approach to clustering, as its representations naturally induce groupings of data points and features.

It is known that optimizing NMF under the orthogonality constraint $\mathbf{W}^T\mathbf{W} = \mathbf{I}$, i.e., solving:

$$\min_{\mathbf{W},\mathbf{H};\mathbf{W}_{i,j}\geq 0 \forall i,j, \mathbf{H}_{i,j}\geq 0 \forall i,j, \mathbf{W}^T\mathbf{W}=\mathbf{I}} \|\mathbf{D} - \mathbf{W}\mathbf{H}\|_F^2 \tag{10}$$

is equivalent to the k-Means problem (assuming positive data) [17].

As in the case of standard clustering, a Khatri-Rao variant of NMF asks to solve:

$$\min_{\substack{\mathbf{W}_t,\mathbf{H}_t \\ \mathbf{W}_{t_{i,j}}\geq 0,\ \mathbf{H}_{t_{i,j}}\geq 0,\ \forall t,i,j \\ \mathbf{W}_t^T\mathbf{W}_t=\mathbf{I}}} \Big\| \mathbf{D} - (\mathbf{W}_1\mathbf{H}_1) \oplus (\mathbf{W}_2\mathbf{H}_2) \oplus \cdots \oplus (\mathbf{W}_p\mathbf{H}_p) \Big\|_F^2. \tag{11}$$

Algorithms that solve the optimization problem in Equation (10) have been proposed in the literature [5]. Studying their extension to the Khatri-Rao clustering paradigm by addressing the optimization problem in Equation 11, is an exciting direction of future work.

D.1 Illustrative experiments.

To conclude, we carry out an illustrative experiment to demonstrate empirically the benefits that Khatri-Rao clustering can provide to centroid-based clustering via matrix factorization. This experiment is not intended to introduce a new state-of-the-art clustering method based on matrix decomposition, but to demonstrate the practical advantages that the Khatri-Rao paradigm can potentially offer in this context through a simple approach.

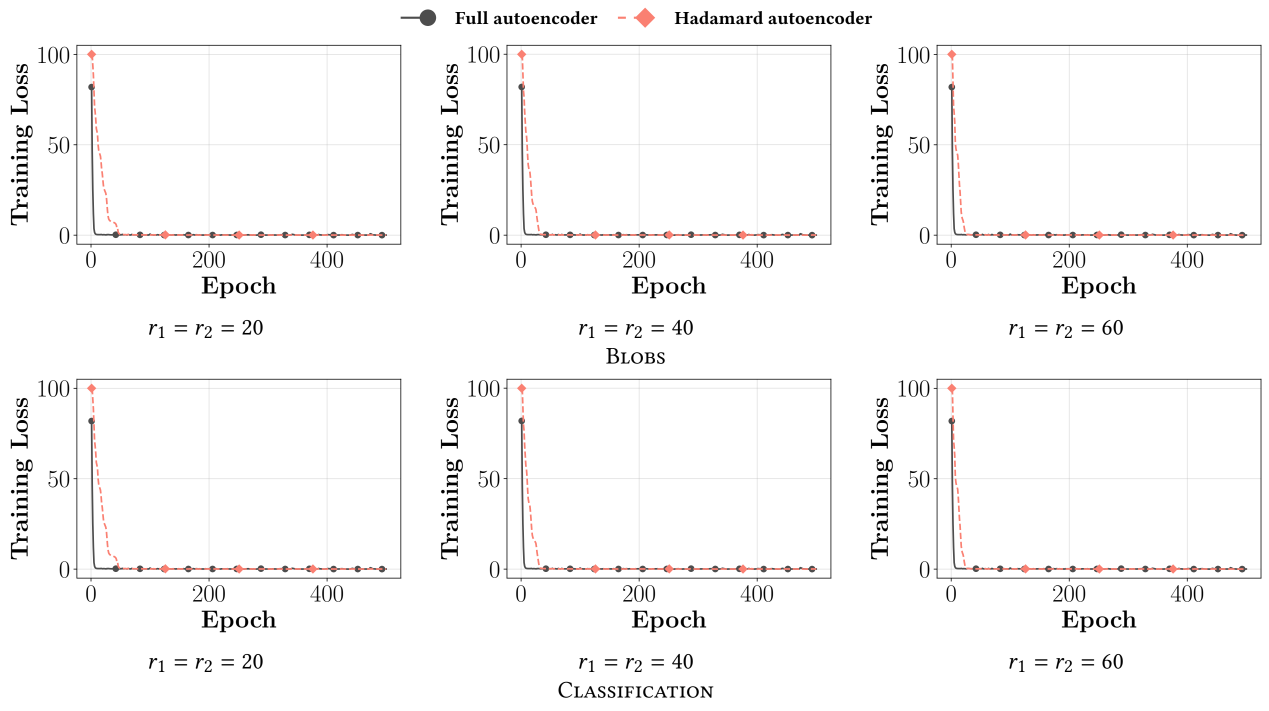

Figure 15: Experiments on Blobs and Classification datasets. Training loss decay for the unconstrained autoencoder and the autoencoder with weights reparameterized as the Hadamard product of two factors with ranks r_1 and r_2 across 500 epochs. We set $r_1 = r_2$, and we show results for three different choices of r_1 and r_2 (20, 40 and 60).

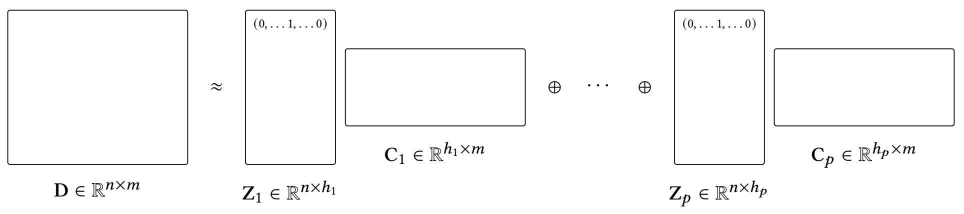
$\approx$
$(0,\dots 1,\dots 0)$
$\oplus \cdots \oplus$
$\mathbf{D} \in \mathbb{R}^{n\times m}$
$\mathbf{Z}_1 \in \mathbb{R}^{n\times h_1}$
$\mathbf{C}_1 \in \mathbb{R}^{h_1\times m}$
$\mathbf{Z}_p \in \mathbb{R}^{n\times h_p}$
$\mathbf{C}_p \in \mathbb{R}^{h_p\times m}$

Figure 16: Diagram representing Khatri-Rao k-Means as matrix decomposition.

Equations (8) and (9) express the k-Means and Khatri-Rao k-Means problems in terms of matrix factorization, respectively. In this illustrative experiment, we address such problems via gradient-based optimization, relying on the Pytorch software for automatic differentiation. In particular, we optimize the following loss function:

$$\sum_{i=1}^{n}\sum_{j=1}^{k} p_{i,j}\|\mathbf{D}_i - \mathbf{C}_j\|_F^2,$$

where $p_{i,j}$ denotes the continuous soft assignment of the i-th data point to the j-th cluster. Such soft assignment is contained in the rows of matrix $\mathbf{Z}$ in standard clustering and in the rows of matrices $\mathbf{Z}_i$ in Khatri-Rao clustering. For both standard and Khatri-Rao clustering, we run the optimization for 5000 epochs using the Adam optimizer, with gradients computed over the entire dataset at each epoch. To obtain one-hot assignment vectors, we initialize the matrices $\mathbf{Z}$ and $\mathbf{Z}_i$ at random, and we apply the softmax function to each row. During the optimization, the temperature of the softmax function is linearly annealed from 1 to 0.1 to progressively sharpen the assignments until they approach one-hot vectors. The learning rate is also annealed at the same time from 10^{-1} to 5×10^{-3}. More specifically, both the learning rate and temperature follow linear schedules of the form $(1 - a_t)l_{start} + a_t l_{end}$, where l_{start} and l_{end}, are either the initial and final learning rate or temperature and a_t increases linearly over epochs. To improve performance, we initialize the centroids (stored in rows of matrix $\mathbf{C}$ for standard clustering and as combinations of rows of matrices $\mathbf{C}_i$ for Khatri-Rao clustering) uniformly in the range of the data.

Upon termination of the 5000 epochs, we compute the hard inertia of the solution clustering by assigning each data point to the cluster with the highest assignment score. Because of the temperature annealing, highest assignment scores are typically close to 1 while the others approach 0.

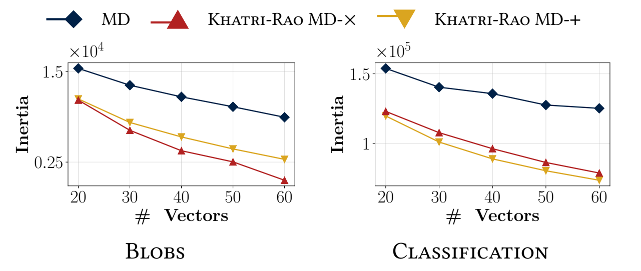

Figure 17: Experiments on Blobs and Classification datasets. Inertia by number of vectors used to represent centroids for a simple approach to standard clustering via matrix decomposition (MD) and its Khatri-Rao variant with sum aggregator (Khatri-Rao MD-+) and product aggregator (Khatri-Rao MD-×). The y-axis is on log scale.

The results of the described experiment for the Blobs and Classification datasets translated to have positive values are displayed in Figure 17. Here, we compare clustering via standard matrix decomposition (for short MD) and its Khatri-Rao clustering with sum aggregator (for short Khatri-Rao MD-+) and product aggregator (for short Khatri-Rao MD-×), as the number of vectors used to represent centroids by both approaches varies in $\{20, 30, 40, 50, 60\}$. The results of the experiment indicate that the matrix decomposition approach to standard clustering incurs higher inertia than its Khatri-Rao counterpart. More specifically, standard matrix decomposition exhibits an inertia that isn at least 31% and 82% higher than that achieved by a Khatri-Rao variant. The advantages given by matrix decomposition based on Khatri-Rao clustering also tend to increase with the number of vectors used to represent centroids.

Appendix E PROOFS

In this section, we collect the proofs of the propositions stated in the paper.

Proof of Proposition 6.1.

Proof. We illustrate the proof for the product aggregator. Consider the j-th protocentroid in the first set of protocentroids. The optimal update for this protocentroid satisfies:

$$\boldsymbol{\theta}_1^j = \arg\min_{\boldsymbol{\theta}} \sum_{l=1}^{h_2} \sum_{\mathbf{x}\in C_{j,l}} (\mathbf{x} - \boldsymbol{\theta} \odot \boldsymbol{\theta}_2^l)^2.$$

Therefore $\boldsymbol{\theta}_1^j$ can be found by computing the gradient of this sum of squared differences with respect to $\boldsymbol{\theta}$ and equating it to the zero vector $\mathbf{0}$. Doing so, one obtains:

$$-2\sum_{l=1}^{h_2} \sum_{\mathbf{x}\in C_{j,l}} (\mathbf{x} - \boldsymbol{\theta} \odot \boldsymbol{\theta}_2^l)\boldsymbol{\theta}_2^l = \mathbf{0},$$

which holds if and only if:

$$\boldsymbol{\theta}_1^j = \frac{\sum_{l=1}^{h_2} \sum_{\mathbf{x}\in C_{j,l}} \mathbf{x} \odot \boldsymbol{\theta}_2^l}{\sum_{l=1}^{h_2} |C_{j,l}| \boldsymbol{\theta}_2^l \odot \boldsymbol{\theta}_2^l}.$$

The derivations for the second set of protocentroids are symmetric, and the proof for the sum aggregator is also similar. □

Proof of Proposition 8.1.

Proof. Assuming p protocentroid sets of equal size and exact budget usage, each set contains $h = \frac{b}{p}$ protocentroids, so the total number of centroids that can be represented is:

$$\left(\frac{b}{p}\right)^p.$$

Maximizing this function over $p > 0$ is equivalent to maximizing its logarithm:

$$p \log\left(\frac{b}{p}\right) = p \log b - p \log p,$$

where log denotes the natural logarithm. Differentiating with respect to p yields:

$$\log\left(\frac{b}{p}\right) - 1,$$

which is positive for $p < \frac{b}{e}$ and negative for $p > \frac{b}{e}$. Moreover, the second derivative:

$$-\frac{1}{p} < 0 \qquad (p > 0)$$

shows that the log-objective is strictly concave, and therefore attains a unique maximum at $p = \frac{b}{e}$ in the continuous domain.

When restricting to the divisor-only setting, p is required to be an integer that exactly divides b, so that $h = \frac{b}{p}$ is an integer. Since the objective is strictly increasing for $p < \frac{b}{e}$ and strictly decreasing for $p > \frac{b}{e}$, among all such admissible values the maximum is attained at one of the two values of p that are immediately below or above $\frac{b}{e}$ (when each exists). All other admissible divisors are necessarily further away from $\frac{b}{e}$, and therefore yield a strictly smaller objective value. □

Proof of Proposition 8.2.

Proof. We present the proof for the product aggregator. However, extension to different aggregator functions is straightforward.

First, to see why it must be $p^* \geq \log_{h_{min}} k$, notice that, if $p^* < \log_{h_{min}} k$ then $h_{min}^{p^*} < k$, and $h_{min}^{p^*}$ is the maximum number of centroids that can be represented using p^* sets of at least h_{min} protocentroids. As regards the other inequality, to prove that $p^* \leq \lceil \frac{k}{h_{min}-1} \rceil$, it is sufficient to construct an example where all sets have a protocentroid with all entries equal to 1 and the remaining at least $h_{min} - 1$ protocentroids that are equal to centroids. Different sets contain different centroids. In the illustrated scenario, each set of protocentroids represents exactly at least $h_{min} - 1$ centroids. Thus, $\lceil \frac{k}{h_{min}-1} \rceil$ sets of protocentroids can always represent k centroids. □

REFERENCES

[1] Kamal Berahmand, Mehrnoush Mohammadi, Razieh Sheikhpour, Yuefeng Li, and Yue Xu. 2024. WSNMF: Weighted Symmetric Nonnegative Matrix Factorization for attributed graph clustering. *Neurocomputing* 566 (2024), 127041.
[2] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. 2004. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences* 101, 12 (2004), 4164–4169.
[3] Zitai Chen, Chuan Chen, Zibin Zheng, and Yi Zhu. 2019. Tensor Decomposition for Multilayer Networks Clustering. In *AAAI*. AAAI Press, 3371–3378.
[4] Peter A. Chew, Brett W. Bader, Tamara G. Kolda, and Ahmed Abdelali. 2007. Cross-language information retrieval using PARAFAC2. In *KDD*. ACM, 143–152.

[5] Seungjin Choi. 2008. Algorithms for orthogonal nonnegative matrix factorization. In *2008 ieee international joint conference on neural networks (ieee world congress on computational intelligence)*. IEEE, 1828–1832.

[6] Martino Ciaperoni, Aristides Gionis, and Heikki Mannila. 2024. The Hadamard decomposition problem. *Data Mining and Knowledge Discovery* (2024), 1–42.

[7] Chris Ding and Xiaofeng He. 2004. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*. 29.

[8] Marek Gagolewski. 2022. A framework for benchmarking clustering algorithms. *SoftwareX* 20 (2022), 101270.

[9] Stephan Günnemann, Ines Färber, Matthias Sebastian Rüdiger, and Thomas Seidl. 2014. SMVC: semi-supervised multi-view clustering in subspace projections. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 253–262. https://doi.org/10.1145/2623330.2623734

[10] Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. 2021. FedPara: Low-rank Hadamard Product for Communication-Efficient Federated Learning. In *International Conference on Learning Representations*.

[11] George Karypis, Eui-Hong Han, and Vipin Kumar. 1999. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer* 32, 8 (1999), 68–75. https://doi.org/10.1109/2.781637

[12] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. 2023. The UCI machine learning repository.

[13] Mikhail Khodak, Neil A. Tenenholtz, Lester Mackey, and Nicolò Fusi. 2021. Initialization and Regularization of Factorized Neural Layers. In *ICLR*. OpenReview.net.

[14] Tamara G. Kolda and Brett W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (2009), 455–500.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[16] Daniel Lee and H Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems* 13 (2000).

[17] Tao Li and Cha-charis Ding. 2018. Nonnegative matrix factorizations for clustering: A survey. *Data Clustering* (2018), 149–176.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] Matteo Ruffini, Ricard Gavaldà, and Esther Limon. 2017. Clustering Patients with Tensor Decomposition. In *MLHC (Proceedings of Machine Learning Research, Vol. 68)*. PMLR, 126–146.

[20] Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. 267–273.