

Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters

Sebastian Salwig^{1,2,†}

SEBASTIAN.SALWIG@UIBK.AC.AT

Till Kahlke^{2,†,*}

TILL.KAHLKE@UIBK.AC.AT

Florian Hirschberger¹

FLORIAN.HIRSCHBERGER@UOL.DE

Dennis Forster³

DENNIS.FORSTER@FB3.FRA-UAS.DE

Jörg Lücke²

JOERG.LUECKE@UIBK.AC.AT

[†]joint first authorship, ^{*}corresponding author

¹*Machine Learning Lab, DMPA, Faculty VI
Carl von Ossietzky University Oldenburg
26129 Oldenburg, Germany*

²*Artificial Intelligence Lab, Department of Computer Science
Faculty of Mathematics, Computer Science and Physics
Innsbruck Research Center for Artificial Intelligence
University of Innsbruck, 6020 Innsbruck, Austria*

³*Data Analytics and AI, Faculty 3
Frankfurt University of Applied Sciences
60318 Frankfurt am Main, Germany*

Editor: Mingyuan Zhou

Abstract

Gaussian Mixture Models (GMMs) range among the most frequently used models in machine learning. However, training large, general GMMs becomes computationally prohibitive for data sets that have many data points N of high-dimensionality D . For GMMs with arbitrary covariances, we here derive a highly efficient variational approximation, which is then integrated with mixtures of factor analyzers (MFAs). For GMMs with C components, our proposed algorithm substantially reduces runtime complexity from $\mathcal{O}(NCD^2)$ per iteration to a complexity scaling linearly with D and sublinearly with NC . In numerical experiments, we first validate that the complexity reduction results in a sublinear scaling for the entire GMM optimization process. Second, we show on large-scale benchmarks that the sublinear algorithm results in speed-ups of an order-of-magnitude compared to the state-of-the-art. Third, as a proof of concept, we finally train GMMs with over 10 billion parameters on about 100 million images, observing training times of less than nine hours on a single state-of-the-art CPU. Finally, and fourth, we demonstrate the effectiveness of large-scale GMMs on the task of zero-shot image denoising, where sublinear training results in state-of-the-art denoising times while competitive denoising performance is maintained.

Keywords: Gaussian mixture models, mixtures of factor analyzers, variational optimization, density estimation, expectation maximization

1. Introduction

In machine learning and data science, Gaussian mixture models (GMMs) are widely used and well-established tools. They are a canonical approach to clustering (e.g., McLachlan and Peel, 2000), can provide valuable insight into data set structures (e.g., Bishop, 2006), or are used as an integral part in conjunction with other approaches for a range of different tasks (e.g., Zoran and Weiss, 2011; Tian et al., 2019; Bouguila and Fan, 2020; Robin and Scrucca, 2023).

One reason for their widespread use is the ability of GMMs to flexibly approximate data densities in potentially high-dimensional data spaces. Any task accomplished by a parametric data density model can, in principle, be addressed using a sufficiently large-scale and sufficiently optimized GMM. Such task generality is possible as GMMs are universal probability density estimators, i.e., they can approximate data densities arbitrarily well (e.g., Parzen, 1962; Escobar and West, 1995; Maz’ya and Schmidt, 1996; Zeevi and Meir, 1997; Li and Barron, 1999).

In the density modeling context, flexible component parametrization and the number of components are crucial for approximation quality (e.g., Zeevi and Meir, 1997; Li and Barron, 1999). However, if GMMs are applied to the large-scale data sets currently used in data science and machine learning, large numbers of components C get combined with many data points N of potentially high dimensions D . In such settings, optimizing general GMMs quickly becomes computationally infeasible. For instance, the runtime cost for executing a single iteration of conventional expectation maximization (EM; Dempster et al., 1977) scales with $\mathcal{O}(NCD^2)$ for general GMMs, making optimization of large-scale models very time-consuming or impractical. To address the limited scalability of conventional GMM optimization, several research directions have been pursued. Each research line discussed in the following aims at reducing the computational complexity.

A common approach is to constrain the covariances by using diagonal covariance matrices (Hirschberger et al., 2022; Exarchakis et al., 2022), which do not model correlations within components. Diagonal covariance matrices reduce the cost of one EM iteration from $\mathcal{O}(NCD^2)$ to $\mathcal{O}(NCD)$. However, such constraints make GMMs much less flexible, potentially impacting their ability to efficiently approximate data densities. Another contribution (Asheri et al., 2021) also constrains covariances but with the main focus on allowing more components C for the same amount of data. This work does not focus on improving scalability compared to conventional GMM optimization. Further approaches (that do address the quadratic scaling with D) include geometrically-oriented approaches (Elkan, 2003; Cheng et al., 1984; Bei and Gray, 1985), random projections (Chan and Leung, 2017), or dimensionality reduction approaches (Bouveyron et al., 2007; Richardson and Weiss, 2018; Hertrich et al., 2022; Liu et al., 2022).

Still another line of research to reduce the optimization cost of GMMs aims at reducing the number of data points. Such reduced data sets are known as coresets (e.g., Lucic et al., 2018; Har-Peled and Mazumdar, 2004; Feldman et al., 2011) and replace the set of N original data points by a smaller weighted set of N' data points. Using the weights from a corresponding coreset algorithm, computational efforts only scale with N' . Coresets

have been used for GMMs with diagonal covariances (Hirschberger et al., 2022; Exarchakis et al., 2022). However, with more general covariances, a reduction to fewer data points is often undesirable, because in high dimensions sufficiently many data points per component are required to appropriately estimate correlations. Other methods that do not directly reduce the number of data points but aim at reducing the computational cost in N include mini-batching (e.g., Nguyen et al., 2020; Sculley, 2010), training on separate subsets of the data set (e.g., Liu et al., 2024), or by hierarchical training schedules (e.g., Richardson and Weiss, 2018).

In contrast to approaches mentioned above, we here aim at decisively reducing computational complexity using variational techniques, while maintaining as flexible as possible GMMs. The main contributions of this work can be summarized as follows:

- (A) We derive a truncated variational optimization method (cf. Lücke and Eggert, 2010; Drefs et al., 2022) applicable to GMMs with arbitrary covariance matrices.
- (B) We introduce a highly efficient learning algorithm based on mixtures of factor analyzers (MFAs, cf. Ghahramani and Hinton, 1996; McLachlan et al., 2003; Richardson and Weiss, 2018), enabling the application of GMMs with billions of parameters to very large-scale data sets.

Additionally, we make use of advanced seeding approaches (e.g., Arthur and Vassilvitskii, 2007; Bachem et al., 2016b,a; Fränti and Sieranoja, 2019). Seeding techniques improve optimization by selecting well-suited initial component centers. But contributions (A) and (B), which jointly define the actual parameter optimization procedure (after initialization), will be the main focus of this work. Our approach enables training of GMMs at scales that were previously considered computationally infeasible.

Regarding research contribution (A), variational optimization is generally used to reduce optimization complexity (Neal and Hinton, 1998; Jordan et al., 1999). Among the most common variational approaches are factored distributions (a.k.a. mean-field distributions) as families of variational distributions (Jordan et al., 1999; Blei and Jordan, 2006, and many more), and Gaussians (Oppen and Archambeau, 2009; Kingma and Welling, 2014). Neither of these variational families is suitable for standard mixture models.¹ But variational approaches have also repeatedly been applied to standard Gaussian mixtures that are the focus of this work (e.g., Neal and Hinton, 1998; Shelton et al., 2017; Forster et al., 2018). In the case of GMMs with diagonal covariances, it has recently been shown that variational optimization can reduce the complexity of one EM iteration from a linear scaling with C to a scaling which is constant w.r.t. C (Hirschberger et al., 2022; Exarchakis et al., 2022). Such complexity reduction has enabled optimization of the, so far, largest scale GMMs with up to 50 000 components and millions of parameters. However, existing procedures for reducing computational complexity have only been applied to GMMs without correlations within clusters (Hirschberger et al., 2022; Exarchakis et al., 2022). The previous derivation of approximate optimization rested on the assumption of Euclidean distances in data space and between component centers. While this assumption can be motivated if the data points

1. Only in the context of fully Bayesian approaches, factorization (of parameter distributions) has been used (e.g., Nasios and Bors, 2006).

within a component *are* uncorrelated, it does not hold for the arbitrary covariances in general GMMs. To address this limitation, we here derive a truncated variational optimization techniques that is directly applicable to GMMs with arbitrary covariance matrices.

Regarding research contribution (B), MFAs flexibly model correlations per component along lower-dimensional hyperplanes. Each component aligns with a hyperplane of $H \leq D$ dimensions, which can be of arbitrary orientation, and which can be different from component to component. As a result, the complexity of an EM step is reduced from $\mathcal{O}(NCD^2)$ to $\mathcal{O}(NCDH)$. For $H = D$, general GMMs are recovered.² For many types of data, however, H can be much smaller than D . Due to their reduced complexity in D , MFAs are applicable to high-dimensional data (e.g., Kock et al., 2022), where they can accomplish tasks usually reserved for neural network approaches (cf. Richardson and Weiss, 2018). However, with current optimization techniques, MFAs still scale approximately linear with NC , which remains their computational bottleneck.

By simultaneously reducing how the optimization complexity depends on NC and how it depends on D , we here enable the optimization of as flexible as possible GMMs at as large scales as possible. The resulting algorithm, that realizes a variational optimization for the flexibly parameterized MFAs, will be referred to as v-MFA.³

The training principles and the derivation of v-MFA are described in Section 2, and its numerical evaluation is described in Section 3.

2. Methods

We will first introduce the class of GMMs we consider, i.e., MFAs, and then derive a variational optimization for MFAs in Sections 2.1 to 2.3. Finally, the implementation of the complete, variational EM algorithm is described in Section 2.4.

2.1 Variational Optimization of MFAs

In the Mixture of Factor Analyzers (MFA) model (e.g., Ghahramani and Hinton, 1996; Richardson and Weiss, 2018), a data point $\mathbf{x} \in \mathbb{R}^D$ is modeled by a hidden mixture component $c \in \{1, \dots, C\}$ and a hidden factor $\mathbf{z} \in \mathbb{R}^H$. Each component with mixing proportion π_c (satisfying $\sum_c \pi_c = 1$) and mean $\boldsymbol{\mu}_c \in \mathbb{R}^D$ represents a factor analyzer (cf. McLachlan and Peel, 2000; McLachlan et al., 2003) with a factor loading matrix $\mathbf{\Lambda}_c \in \mathbb{R}^{D \times H}$. The generative model is given by

$$p(c | \boldsymbol{\Theta}) = \pi_c, \quad p(\mathbf{z} | \boldsymbol{\Theta}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad p(\mathbf{x} | c, \mathbf{z}, \boldsymbol{\Theta}) = \mathcal{N}(\mathbf{x}; \mathbf{\Lambda}_c \mathbf{z} + \boldsymbol{\mu}_c, \mathbf{D}_c), \quad (1)$$

where $\mathcal{N}$ represents a multivariate Gaussian distribution, $\mathbf{D}_c := \text{diag}(\sigma_{c,1}^2, \dots, \sigma_{c,D}^2) \in \mathbb{R}^{D \times D}$ is a diagonal matrix containing independent noise, and $\boldsymbol{\Theta} := \{\pi_{1:C}, \mathbf{\Lambda}_{1:C}, \boldsymbol{\mu}_{1:C}, \mathbf{D}_{1:C}\}$ denotes all model parameters.

2. The GMM is overparameterized in this case.

3. Code is available at: <https://github.com/variational-sublinear-clustering/vamm>

The MFA model can also be considered as a GMM with low-rank covariance matrices $\Sigma_c = \Lambda_c \Lambda_c^\top + D_c$. In this case, the generative model is expressed as

$$p(c | \Theta) = \pi_c, \quad p(\mathbf{x} | c, \Theta) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Lambda_c \Lambda_c^\top + D_c). \quad (2)$$

The reformulation results from marginalization over $\mathbf{z}$ using Gaussian identities.

To fit the MFA model to a given data set $\mathbf{x}_{1:N}$, we seek parameters $\Theta^* = \operatorname{argmax}_{\Theta} \mathcal{L}(\Theta)$ that optimize the log-likelihood given by

$$\mathcal{L}(\mathbf{x}_{1:N}; \Theta) = \sum_{n=1}^N \log \left(\sum_{c=1}^C \pi_c \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_c, \Lambda_c \Lambda_c^\top + D_c) \right). \quad (3)$$

Direct log-likelihood optimization is usually difficult. Instead, efficient algorithms often employ approaches such as Expectation Maximization (EM) or variational approximations of EM (Dempster et al., 1977; Neal and Hinton, 1998; Jordan et al., 1999). Variational approaches optimize the free energy, which is a lower bound of the log-likelihood (also known as the evidence lower bound—ELBO). The free energy is iteratively optimized by computing expectation values of latent variables in the E-step and updating the model parameters Θ in the M-step. In this study, we employ a variational version of the EM algorithm, which uses truncated posterior distributions (e.g., Lücke and Eggert, 2010; Shelton et al., 2017; Drefs et al., 2022) as its family of variational distributions. Concretely, we use variational distributions $q_n(c; \tilde{\Theta})$ defined by

$$\forall n = 1, \dots, N : \quad q_n(c; \tilde{\Theta}) := q(c; \mathbf{x}_n, \mathcal{K}_n, \tilde{\Theta}) = \frac{p(c, \mathbf{x}_n | \tilde{\Theta})}{\sum_{\tilde{c} \in \mathcal{K}_n} p(\tilde{c}, \mathbf{x}_n | \tilde{\Theta})} \delta(c \in \mathcal{K}_n), \quad (4)$$

where $\delta(c \in \mathcal{K}_n)$ is equal to 1 if $c \in \mathcal{K}_n$ and 0 otherwise (also known as Iverson bracket), and $\mathcal{K}_n$ denotes those component indices for which the variational distribution $q_n(c; \tilde{\Theta})$ is non-zero. Throughout this work, we will assume that the size of any set $\mathcal{K}_n$ is restricted to $C' \leq C$ indices, i.e., $|\mathcal{K}_n| = C'$ for all n .

If we use truncated distributions (Equation 4), we obtain as variational free energy objective (see Appendix A for details)

$$\mathcal{F}(\mathbf{x}_{1:N}; \mathcal{K}, \tilde{\Theta}, \Theta) := \sum_{n=1}^N \left(\sum_{c=1}^C q_n(c; \tilde{\Theta}) \log (\pi_c \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_c, \Lambda_c \Lambda_c^\top + D_c)) + \mathcal{H}[q_n] \right), \quad (5)$$

where $\mathcal{K}$ is the collection of all index sets $\mathcal{K}_{1:N}$, and $\mathcal{H}[q_n] = -\mathbb{E}_{q_n}[\log q_n(c; \tilde{\Theta})]$ is the Shannon entropy. A detailed derivation is given in the appendix of Lücke (2019). The ‘hard’ zeros used by the truncated distributions mean that we effectively only have to evaluate C' non-zero summands per data point. This reduces the computational complexity of one M-step from linearly scaling with C to a linear scaling with C' .

The optimal values of the variational parameters $\tilde{\Theta}$ are the current values of the model parameters Θ , which can be shown in general (Lücke, 2019) and, therefore, also applies for

the MFA model. For $\tilde{\Theta} = \Theta$, the free energy simplifies to (see Appendix A for details)

$$\mathcal{F}(\mathcal{K}, \Theta) := \mathcal{F}(\mathbf{x}_{1:N}; \mathcal{K}, \Theta, \Theta) = \sum_{n=1}^N \log \left(\sum_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta) \right). \quad (6)$$

As could be deduced from Equation (6), optimizing the MFA model using variational EM requires to repeatedly evaluate joint probabilities $p(c, \mathbf{x}_n | \Theta)$ or, equivalently, log-joints. In previous work (Hirschberger et al., 2022; Exarchakis et al., 2022), the use of GMMs with diagonal covariance matrices enabled the evaluation of a single log-joint in $\mathcal{O}(D)$. However, in this work, the goal is to model correlations, which substantially increases the computational complexity to $\mathcal{O}(D^2)$ when all correlations are considered. By modeling data distributions along lower-dimensional hyperplanes, the MFA model reduces the complexity to $\mathcal{O}(DH)$ while preserving arbitrary correlations along its hyperplanes. Further details on the computational techniques employed for the MFA model, as well as the derivations of the parameter updates, can be found in Appendices A.1 and A.2, respectively. Further details on the relation of the variational objective Equation (5) to the log-likelihood objective Equation (3) can be found in Appendix A.3.

2.2 Efficient Partial Variational E-steps

As with other variational approaches, the crucial challenge is to derive a computationally efficient E-step. To do so for the MFA model, we follow here a strategy similar to previous work (Hirschberger et al., 2022; Exarchakis et al., 2022), which has derived efficient partial variational E-steps focusing on diagonal covariance matrices.

We start by noting that the joint probabilities defined by the MFA model play a central role in the optimization of the free energy. This role is underlined by the following proposition:

Proposition 1: Consider the joint probability $p(c, \mathbf{x}_n | \Theta)$ defined by the MFA generative model in Equation (2), and the free energy $\mathcal{F}$ for index sets $\mathcal{K}$. If we replace a component $c \in \mathcal{K}_n$ by a component $\tilde{c} \notin \mathcal{K}_n$ (for an arbitrary n), then the free energy increases if and only if $p(\tilde{c}, \mathbf{x}_n | \Theta) > p(c, \mathbf{x}_n | \Theta)$.

Proposition 1 is a special case of a general result for truncated variational distributions (Lücke, 2019). It therefore applies to any mixture model and to MFA as a special case. For completeness, we provide the proof for general mixture models in Appendix A.4.

Proposition 1 then leads to an equivalent definition of the optimal variational parameters $\mathcal{K}_n$ as in previous work (Hirschberger et al., 2022; Exarchakis et al., 2022). $\mathcal{K}_n^{\text{opt}}$ is given by

$$\mathcal{K}_n^{\text{opt}} = \{c \mid p(c, \mathbf{x}_n | \Theta) \text{ is among the } C' \text{ largest joints}\}, \quad (7)$$

i.e., the optimal sets $\mathcal{K}_n^{\text{opt}}$ contain those C' components with the highest joint probabilities (also see Shelton et al., 2017; Forster et al., 2018). However, finding the optimal components would require the evaluation of C joints per data point, i.e., NC joint evaluations in total. Hence, finding the optimal $\mathcal{K}_n$ given by Equation (7) would require the same number of

joint evaluations as a full E-step. To also reduce the computational complexity of the E-step, we consequently seek a procedure similar to Hirschberger et al. (2022) and Exarchakis et al. (2022), that builds upon partial variational E-steps, i.e., we seek an increase of the free energy instead of its maximization. In virtue of Proposition 1, we can increase the free energy by identifying components with *higher* joints rather than finding the C' components with the *highest* joints. This can be accomplished efficiently by evaluating only a subset of all joint probabilities during each E-step, which reduces the computational load compared to a full E-step. Concretely, similar to Hirschberger et al. (2022), we introduce a set $\mathcal{S}_n$, referred to as the search space,⁴ that includes candidate components $\tilde{c}$, which may replace a $c \in \mathcal{K}_n$ to increase the free energy for each data point $\mathbf{x}_n$. The size of $\mathcal{S}_n$ is upper-bounded, i.e., we demand for all n that $|\mathcal{S}_n| \leq S$. Here, S will be chosen to be larger than C' but significantly smaller than C (i.e., $C' < S \ll C$). Instead of finding the *optimal* sets $\mathcal{K}_n^{\text{opt}}$, we now partially optimize each $\mathcal{K}_n$ using the search space $\mathcal{S}_n$ of a given $\mathbf{x}_n$. Given $\mathcal{S}_n$, the updated $\mathcal{K}_n$ is defined by

$$\mathcal{K}_n = \{c \mid p(c, \mathbf{x}_n \mid \Theta) \text{ is among the } C' \text{ largest joints for all } c \in \mathcal{S}_n\}. \quad (8)$$

The index set $\mathcal{K}_n$ of Equation (8) can consequently be determined by evaluating all joints with $\mathbf{x}_n$ and all $c \in \mathcal{S}_n$ as arguments.

We will define each search space $\mathcal{S}_n$ to contain $\mathcal{K}_n$ as subset ($\mathcal{K}_n \subset \mathcal{S}_n$), which guarantees that the update of $\mathcal{K}_n$ according to Equation (8) never decreases the free energy $\mathcal{F}(\mathcal{K}, \Theta)$. However, this condition on the search spaces $\mathcal{S}_n$ is not sufficient to warrant an efficient increase of the free energy. For high efficiency, replacement candidates $\tilde{c} \in \mathcal{S}_n \setminus \mathcal{K}_n$ have to be likely to result in larger joints, i.e., it has to be sufficiently likely that $\tilde{c} \in \mathcal{S}_n$ with $\tilde{c} \notin \mathcal{K}_n$ and $c \in \mathcal{K}_n$ satisfy $p(\tilde{c}, \mathbf{x}_n \mid \Theta) > p(c, \mathbf{x}_n \mid \Theta)$.

A first trivial option to define $\mathcal{S}_n$ would be to use all $c \in \mathcal{K}_n$ as members of $\mathcal{S}_n$, and to then add component indices that are uniformly sampled from $\{1, \dots, C\}$. However, as C increases, the probability of finding a new component $\tilde{c}$ with high joint $p(\tilde{c}, \mathbf{x}_n \mid \Theta)$ becomes increasingly small. Therefore, we seek an approach that suggests components $\tilde{c}$ that are more likely to increase the free energy than such a blind random search.

2.3 Construction of the Search Space for Guided Search

In order to allow for a more guided search, we will define the search spaces $\mathcal{S}_n$ to contain component indices likely to improve the free energy while being still efficiently computable. We remain with a data point centric approach and use a bootstrapping strategy to continuously improve the search spaces $\mathcal{S}_n$ together with the sets $\mathcal{K}_n$. In line with previous research (Hirschberger et al., 2022), we define the search spaces $\mathcal{S}_n$ by making use of (data point independent) sets g_c , that contain replacement candidates for c . However, in this work, the definition of these sets will be introduced in a novel manner to account for GMMs with arbitrary covariances. The components $\tilde{c} \in g_c$ can, for now, be thought of as components estimated to be ‘similar’ to component c , but we will only precisely define the g_c further below. The intuition behind using g_c sets goes as follows: if a data point $\mathbf{x}$ lies in the vicinity of a component c , and component c is ‘similar’ to another component $\tilde{c}$ then, by

4. The search space is denoted as $\mathcal{G}_n$ in Hirschberger et al. (2022).

transitivity, $\mathbf{x}$ is also likely to be in the vicinity of $\tilde{c}$. Using the sets g_c (and one component sampled uniformly at random), a search space $\mathcal{S}_n$ for a given data point is defined as the following union:

$$\mathcal{S}_n := \bigcup_{c \in \mathcal{K}_n} g_c \cup \{c'\} \quad \text{with} \quad c' \sim \mathcal{U}\{1, C\}. \quad (9)$$

Note that without the random component, there is no theoretical guarantee that the optimization process can find the optimal variational parameters $\mathcal{K}_n^{\text{opt}}$ according to Equation (7) even in the limit of infinitely many iterations.

To ensure that the size of the search space is upper bounded, we restrict the maximum size of the sets g_c to G . The maximum size of a search space $\mathcal{S}_n$ is thus given by $S = |\mathcal{K}_n| |g_c| + 1 = C'G + 1$. In general, however, the index sets g_c may intersect, leading to smaller $\mathcal{S}_n$.

The construction of the search space by Equation (9) can be conceptualized as a search tree structure. Specifically, we build $\mathcal{S}_n$ by first considering all components in $\mathcal{K}_n$, which contains the most relevant components identified so far for $\mathbf{x}_n$. Next, for each component $c \in \mathcal{K}_n$, we consider the set of components in g_c , that provides replacement candidates for component c . Figure 1 offers an illustration of the sets g_c and the construction of $\mathcal{S}_n$. Figure 1A shows a schematic representation, Figure 1B visualize $\mathcal{K}_n$ and g_c , and Figure 1C illustrates $\mathcal{S}_n$ in data space. Figure 1 also demonstrates that $|\mathcal{S}_n|$ is larger than $|\mathcal{K}_n|$, but significantly smaller than C .

It remains to define the sets g_c . In previous work (Hirschberger et al., 2022), the sets g_c were updated based on Euclidean neighborhoods. Concretely, a set g_c contained components $\tilde{c}$ with small Euclidean distances $\|\boldsymbol{\mu}_c - \boldsymbol{\mu}_{\tilde{c}}\|$ to the component c (with $\boldsymbol{\mu}_c$ and $\boldsymbol{\mu}_{\tilde{c}}$ denoting component centers). These component-to-component distances were then approximated using component-to-datapoint distances (for further details, see Hirschberger et al., 2022).

In our case, however, covariance matrices are not restricted to the uncorrelated case. The general and potentially strong correlations we seek to model can strongly change proximity relations among components as well as between data points and components, which makes previous Euclidean distance metrics (Hirschberger et al., 2022; Exarchakis et al., 2022) unsuitable for finding relevant components for a given component. Figure 2 highlights that for general covariance matrices, small Euclidean distances no longer correspond to large joint values.

Instead of finding components similar to component c using Euclidean distances, we measure component similarities via the Kullback-Leibler (KL) divergence. The KL-divergence between the probability distributions of a component c and component $\tilde{c}$ is given by

$$D_{\text{KL}} [p(\mathbf{x} | c, \boldsymbol{\Theta}) || p(\mathbf{x} | \tilde{c}, \boldsymbol{\Theta})] = \mathbb{E}_{p(\mathbf{x} | c, \boldsymbol{\Theta})} \left[\log \frac{p(\mathbf{x} | c, \boldsymbol{\Theta})}{p(\mathbf{x} | \tilde{c}, \boldsymbol{\Theta})} \right]. \quad (10)$$

Although there exists an analytically tractable form of the KL-divergence for Gaussian distributions, evaluating Equation (10) introduces additional computational overhead. Con-

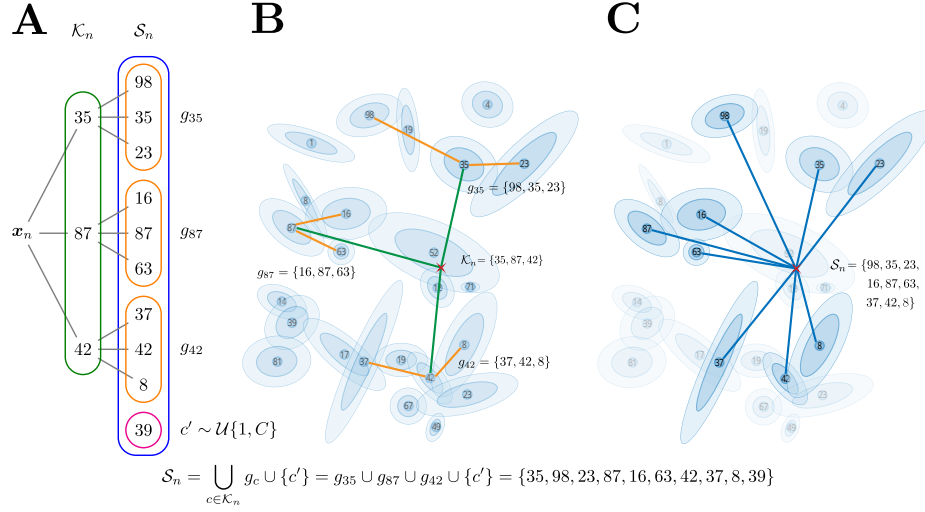


Figure 1: Construction of S_n for a data point x_n . Subfigure **A** shows a schematic representation while **B** visualizes $\mathcal{K}_n$ and g_c , and **C** visualizes S_n in data space. In this example, the set $\mathcal{K}_n = \{35, 87, 42\}$ contains $C' = 3$ component indices for x_n , as indicated by the green box in **A** and green lines in **B**. Each of the three components has a set of neighborhoods g_c that includes the component c itself and two additional component indices, i.e., $|g_c| = G = 3$. Specifically, we have $g_{35} = \{98, 35, 23\}$, $g_{87} = \{16, 87, 63\}$ and $g_{42} = \{37, 42, 8\}$. These three sets are highlighted by orange boxes in **A** and orange lines in **B**. The union of the three sets g_c and a randomly added component $c' = 39$ forms the search space $S_n = \{98, 35, 23, 16, 87, 63, 37, 42, 8, 39\}$, as indicated by the blue box in **A** and blue lines in **C**. None of the sets are optimal for the data point x_n yet. The optimization procedure will be explained in the following.

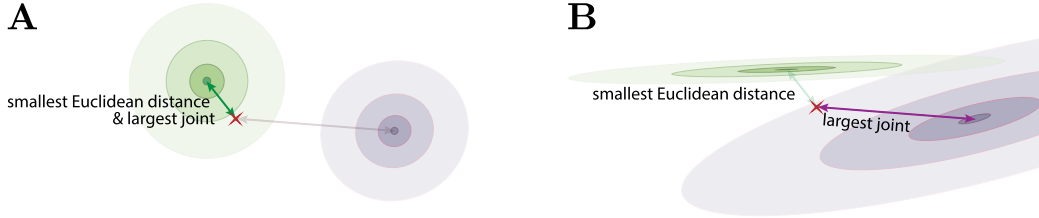


Figure 2: Comparison of Euclidean distances and joints. In **A** and **B**, the component centers are identical and the same data point is marked with a red cross. For isotropic, equally weighted components (**A**), the smallest Euclidean distance correspond to the largest joint between components and the data point. However, for general covariance matrices (**B**), this is no longer the case.

cretely, the divergence would have to be computed for all $(c, \tilde{c})$ pairs, resulting in a complexity of $\mathcal{O}(C^2DH)$ for the MFA model. As we aim at handling large numbers of components efficiently, the quadratic scaling with C quickly becomes a substantial computational bottleneck. Thus, we aim to efficiently estimate the KL-divergence, while still capturing sufficient

information about the KL-divergences for those $(c, \tilde{c})$ pairs that are relevant for the optimization algorithm. We stress that this highly efficient estimation of the KL-divergence for GMMs with general covariances is the crucial step to enable scalability of the whole variational algorithm. While other measures are conceivable, the use of KL-divergences may in our context be a canonical choice.

To achieve an as efficient as possible estimation of Equation (10), we employ a finite sample approximation. Instead of drawing samples from $p(\mathbf{x} | c, \Theta)$, we estimate the KL-divergence by using the data points currently assigned to component c . This gives rise to a partitioning of the data set into the following sets:

$$\mathcal{I}_c = \{n \mid c = c_n\} \quad \text{with} \quad c_n = \operatorname{argmax}_{c' \in \mathcal{K}_n} p(c' | \mathbf{x}_n, \Theta) = \operatorname{argmax}_{c' \in \mathcal{K}_n} p(c', \mathbf{x}_n | \Theta). \quad (11)$$

Given the sets $\mathcal{I}_c$, the KL-divergence between the probability distributions of a component $c \in \mathcal{K}_n$ and a potential candidate $\tilde{c}$ to add to g_c can be estimated as follows (see Appendix A.5 for details)

$$D_{\text{KL}} [p(\mathbf{x} | c, \Theta) \parallel p(\mathbf{x} | \tilde{c}, \Theta)] \approx \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log \frac{p(\mathbf{x}_n | c, \Theta)}{p(\mathbf{x}_n | \tilde{c}, \Theta)} \delta(\tilde{c} \in \mathcal{S}_n) =: D_{c\tilde{c}} \quad (12)$$

where $N_{c\tilde{c}} = \sum_{n \in \mathcal{I}_c} \delta(\tilde{c} \in \mathcal{S}_n)$.

If we disregard the condition $\delta(\tilde{c} \in \mathcal{S}_n)$, the function $D_{c\tilde{c}}$ estimates the KL-divergence using *all* data points in $\mathcal{I}_c$. The condition $\delta(\tilde{c} \in \mathcal{S}_n)$, however, reduces the number of samples to approximate the KL-divergence by considering only those data points for which component $\tilde{c}$ is present in their search spaces. This ensures that no additional computations of $p(\mathbf{x}_n | \tilde{c}, \Theta)$ are required beyond those that anyway have to be performed in a partial E-step (see Section 2.4 for details). As a result, $D_{c\tilde{c}}$ are very efficiently computable, which is the primary motivation for employing this approximation. Due to $\delta(\tilde{c} \in \mathcal{S}_n)$ in Equation (12), the function $D_{c\tilde{c}}$ is defined only for a restricted subset of components $\tilde{c}$. For those $\tilde{c}$ that do not appear in any of the search spaces, (i.e., any $\tilde{c} \notin \bigcup_{n \in \mathcal{I}_c} \mathcal{S}_n$), we set $D_{c\tilde{c}}$ to infinity, formally treating the component $\tilde{c}$ for c as irrelevant.

Equation (12) yields a ranking, where a lower value of $D_{c\tilde{c}}$ indicates a greater similarity of component $\tilde{c}$ to c . Consequently, we can define g_c as the set of those components that result in the G lowest values of $D_{c\tilde{c}}$, i.e.,

$$g_c = \{\tilde{c} \mid D_{c\tilde{c}} \text{ is among the } G \text{ lowest values}\}. \quad (13)$$

A potential drawback of using Equation (12) is that $D_{c\tilde{c}}$ may only provide a coarse estimate of the KL-divergence, depending on the convergence state of the search spaces, the model parameters, etc. (we elaborate in Appendix A.5). Importantly, however, the estimate still captures sufficient information about component similarities in the sense of KL-divergences. Since we only require the values of $D_{c\tilde{c}}$ to rank components $\tilde{c}$, their exact values are of secondary importance.

We note that a relation to KL-divergences was also discussed in Hirschberger et al. (2022). However, the general distance introduced in Hirschberger et al. (2022) was approximated in

such a way that, for isotropic GMMs, it recovers a measure based on Euclidean distances. Further details, along with numerical comparisons to our novel and generalized method, are provided in Appendices A.5.1 and B.2.1. While the here defined method (based on Equation 12) is especially advantageous in the general case with intra-component correlations, we also observed improvements for the uncorrelated case.

2.4 Algorithmic Realization of v-MFA

While Sections 2.1 to 2.3 described the theoretical foundation of the variational learning algorithm for MFA models, we now focus on its algorithmic realization (which we will refer to as v-MFA). Analogously to conventional EM optimization, v-MFA alternates between E-steps and M-steps to update variational parameters and model parameters, respectively. The variational E-step is discussed below, and details on the M-step are provided in Appendix A.2.

Variational E-step: Algorithm 1 shows the partial variational E-step, which comprises four blocks (variables are implicitly initialized with zeros or empty sets, respectively):

- (1) For each data point, the search space $\mathcal{S}_n$ is constructed and a component, drawn from a discrete uniform distribution $\mathcal{U}\{1, C\}$, is added to $\mathcal{S}_n$ (it consequently applies that $|\mathcal{S}_n| \leq C'G + 1$). Subsequently, the joints are evaluated for all $c \in \mathcal{S}_n$. The $\mathcal{K}_n$ sets are then optimized using these joints. Figure 3 visualizes the iterative procedure for $\mathcal{K}_n$ optimization.
- (2) The partition $\mathcal{I}_{1:C}$ of the data set is computed.
- (3) The partition and the previously computed joints are used to determine the sets $g_{1:C}$, utilizing $\log p(\mathbf{x}_n | c, \Theta) = \log p(c, \mathbf{x}_n | \Theta) - \log p(c | \Theta)$. $D_{c\tilde{c}} = \tilde{D}_{c\tilde{c}}/N_{c\tilde{c}}$ will be computed for all $\tilde{c} \in \bigcup_{n \in \mathcal{I}_c} \mathcal{S}_n$ excluding $\tilde{c} = c$, because c will always be included in g_c .
- (4) The variational distributions $q_n(c; \Theta)$ are computed by normalizing of $p(c, \mathbf{x}_n | \Theta)$.

The computational complexity of a single variational E-step is $\mathcal{O}(NSDH)$, dominated by the joint computations in block 1 of Algorithm 1 (assuming $N > C$). Unlike conventional E-steps, which require evaluating NC joint probabilities by considering combinations of all data points with all components, the variational E-step only loops over all data points and their respective search spaces. As a result, it requires at most NS joint evaluations, which is significantly fewer when $S \ll C$. A detailed specification of the complexity is provided in Appendix A.6.

Initialization of model parameters: To preserve the algorithmic complexity of Algorithm 1, initialization methods should exhibit complexities that are comparable to or lower than those of the optimization algorithm itself. The k -means++ seeding method (Arthur and Vassilvitskii, 2007) is not suitable in our context, for instance, because it scales with $\mathcal{O}(NCD)$. Therefore, we employ AFK-MC² (Bachem et al., 2016a) as an efficient initialization method for the means μ_c . To initialize the diagonal variances $\mathbf{D}_{1:C}$, we compute the variance along each dimension across all data points as follows

$$\mathbf{D}_c = \mathbf{D}_{\text{init}} \quad \forall c \quad \text{with } \mathbf{D}_{\text{init}} = \text{diag}(\sigma_1^2, \dots, \sigma_d^2) \text{ and } \sigma_d^2 = \text{var}(x_{1:N,d}),$$

Algorithm 1: Variational E-step

```

for  $n = 1 : N$  do
   $\mathcal{S}_n = \bigcup_{c \in \mathcal{K}_n} g_c$ ;
  1  $\mathcal{S}_n = \mathcal{S}_n \cup \{c\}$  with  $c \sim \mathcal{U}\{1, C\}$ ;
    for  $c \in \mathcal{S}_n$  do
       $\lfloor$  compute joint  $p(c, \mathbf{x}_n | \Theta)$ ;
       $\mathcal{K}_n = \{c \mid p(c, \mathbf{x}_n | \Theta) \text{ is among the } C' \text{ largest joints for all } c \in \mathcal{S}_n\}$ ;


---


  for  $n = 1 : N$  do
    2  $c_n = \operatorname{argmax}_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta)$ ;
     $\mathcal{I}_{c_n} = \mathcal{I}_{c_n} \cup \{n\}$ ;


---


  for  $c = 1 : C$  do
    for  $n \in \mathcal{I}_c$  do
      3 for  $\tilde{c} \in \mathcal{S}_n \setminus \{c\}$  do
         $\tilde{D}_{c\tilde{c}} = \tilde{D}_{c\tilde{c}} + \log p(\mathbf{x}_n | c, \Theta) - \log p(\mathbf{x}_n | \tilde{c}, \Theta)$ ;
         $N_{c\tilde{c}} = N_{c\tilde{c}} + 1$ ;
       $g_c = \{\tilde{c} \mid (\tilde{D}_{c\tilde{c}}/N_{c\tilde{c}}) \text{ is among the } G - 1 \text{ smallest values of all computed } (\tilde{D}_{c\tilde{c}}/N_{c\tilde{c}})\} \cup \{c\}$ ;


---


  for  $n = 1 : N$  do
    4 for  $c \in \mathcal{K}_n$  do
       $q_n(c; \Theta) = p(c, \mathbf{x}_n | \Theta) / \sum_{\tilde{c} \in \mathcal{K}_n} p(\tilde{c}, \mathbf{x}_n | \Theta)$ ;

```

where $x_{n,d}$ is the d -th entry of $\mathbf{x}_n$. For the factor loading matrices $\mathbf{A}_c$, we use uniform random numbers between 0 and 1 to set their values. Initialization methods proposed by Richardson and Weiss (2018) and McLachlan et al. (2003) are computationally too demanding for our specific setting.

Initialization of variational parameters: The initialization of index sets $\mathcal{K}_{1:N}$ and $g_{1:C}$ proceeds as follows: Data points designated as seeds for components, sampled by AFK-MC², ensure that the corresponding component index is contained within the respective set $\mathcal{K}_n$. The set g_c consistently includes index c . Subsequently, $\mathcal{K}_{1:N}$ and $g_{1:C}$ are populated with further indices by sampling uniformly from $\{1, \dots, C\}$ (while ensuring uniqueness of the indices). Following this, $\mathcal{K}_{1:N}$ and $g_{1:C}$ are optimized through initial partial E-steps, while keeping the model parameters fixed. This approach has been previously shown to yield favorable results (Hirschberger et al., 2022). We refer to this procedure as *warm-up*.

Complete v-MFA: The implementation of the complete v-MFA algorithm is outlined in Algorithm 2. Following initialization, the algorithm performs variational E-steps in a *warm-up* phase until the free energy $\mathcal{F}(\mathcal{K}, \Theta)$ has converged. Subsequently, the main training process alternates between variational E-steps and M-steps until $\mathcal{F}(\mathcal{K}, \Theta)$ has converged again. As convergence criterion for both loops, we consider $\mathcal{F}(\mathcal{K}, \Theta)$ to have converged

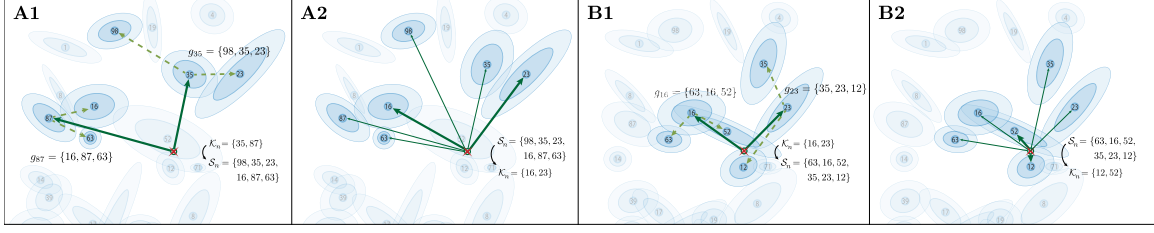


Figure 3: Update of $\mathcal{K}_n$ over two iterations. In **A1**, a red cross marks a data point $\mathbf{x}_n$ with an initial suboptimal $\mathcal{K}_n = \{35, 87\}$ (for simplicity, we use only $C' = 2$ here) and $G = 3$ replacement candidates for each of those components: $g_{35} = \{98, 35, 23\}$ and $g_{87} = \{16, 87, 63\}$. Also these replacement candidates are still suboptimal. Subsequently, in **A2**, the joints $p(c, \mathbf{x}_n | \Theta)$ for all components in the search space $\mathcal{S}_n = g_{35} \cup g_{87} \cup \{c'\} = \{98, 35, 23, 16, 87, 63, 39\}$ are calculated, and the $C' = 2$ components with the largest joints are selected as new $\mathcal{K}_n = \{16, 23\}$. In the following steps of the algorithm, the replacement candidates g_c will be adjusted based on the current $\mathcal{S}_n$, and the model parameters will be updated in the M-step. In **B1** and **B2** the same procedure is repeated in the next iteration to find components $\mathcal{K}_n$ with larger joints. In this example, over two iterations, a total of 14 joints were calculated to find the best final $\mathcal{K}_n$, which is only a fraction of the total number of calculations required had we considered all C components.

Algorithm 2: v-MFA

```

initialize  $\pi_{1:C}, \mu_{1:C}, \Lambda_{1:C}, \mathbf{D}_{1:C}, \mathcal{K}_{1:N}$  and  $g_{1:C}$  (see text for details);
repeat
1 | variational E-step: update  $\mathcal{K}_{1:N}, g_{1:C}$  using Algorithm 1;
   until  $\mathcal{F}(\mathcal{K}, \Theta)$  has converged;

   repeat
2 | variational E-step: update  $\mathcal{K}_{1:N}, g_{1:C}$  using Algorithm 1;
   | variational M-step: update  $\pi_{1:C}, \mu_{1:C}, \Lambda_{1:C}$  and  $\mathbf{D}_{1:C}$  (see Appendix A.2 for
   | details);
   until  $\mathcal{F}(\mathcal{K}, \Theta)$  has converged;
    
```

after iteration t if

$$\frac{\mathcal{F}^{(t)} - \mathcal{F}^{(t-1)}}{\mathcal{F}^{(t-1)}} < \varepsilon, \quad (14)$$

where $\mathcal{F}^{(t)}$ is the free energy $\mathcal{F}(\mathcal{K}, \Theta)$ of Equation (6) after iteration t (cf. Hirschberger et al., 2022). In all numerical experiments we choose the convergence threshold $\varepsilon = 10^{-4}$ unless specifically stated otherwise.

3. Numerical Experiments

To assess the efficiency and effectiveness of the v-MFA algorithm, we conducted numerical experiments organized into four sections. In Section 3.1, we provide a comprehensive comparison of the scaling behavior of v-MFA with increasing component numbers C and compared to conventional EM optimization of MFA models (which we will refer to as em-MFA). Throughout our evaluation, the em-MFA algorithm serves as a canonical baseline. Following the scaling analysis, we evaluate in Section 3.2 the performance and quality of v-MFA relative to em-MFA. We also compare v-MFA to a k -means algorithm combined with factor analysis (cf. Richardson and Weiss, 2018), denoted as k -means+FA. In Section 3.3, we extend the analysis to data set and model sizes where the application of em-MFA is no longer feasible. On such tasks the efficiency of v-MFA translates to its ability to train GMMs at scales that have previously not been reported. Finally, in Section 3.4, we benchmark the performance and efficiency of v-MFA on one specific downstream task: image denoising. The task allows for a comparison of v-MFA to various other algorithms, including self-supervised deep neural networks (DNNs) and variational autoencoder-based approaches as examples of deep generative models. Further details on all numerical experiments are provided in Appendix B.1, and further control experiments are reported in Appendix B.2.

In Sections 3.1 and 3.2, we evaluated v-MFA on four well-established benchmarks for high-dimensional image data, namely CIFAR-10 (Krizhevsky, 2009), CelebA (Liu et al., 2015), EMNIST (Cohen et al., 2017) and SVHN (Netzer et al., 2011) (for details, see Appendix B.1.3). The data sets have frequently been applied in different contexts, and they were used previously in settings similar to the one of interest here (Richardson and Weiss, 2018; Hirschberger et al., 2022; Exarchakis et al., 2022). Image data is commonly used for benchmarking, and is often considered as an example for data points lying close to a data manifold with much lower dimension than the dimensionality of the data. Images are consequently well suited to assess the effectiveness and efficiency of algorithms that optimize MFAs. Moreover, the chosen data sets provide a sufficiently large number of data points to estimate components for the large-scale MFA models we are interested in.

In the numerical experiments, we evaluated the efficiency of v-MFA and corresponding baselines in different settings using two measures for computational costs: the total number of joint probability evaluations (treated as generalized distances) and the wall-clock runtime (cf. Hirschberger et al., 2022; Exarchakis et al., 2022).

The first measure, the total number of joint evaluations, is independent of concrete implementation details and the hardware used. The number of joint evaluations also excludes initialization, parameter updates within the M-steps and auxiliary computations. The second measure, the wall-clock runtime, encompasses all computations, memory management, and other auxiliary routines. Actual improvements in wall-clock runtime are practically relevant, although this measure depends on implementation details and the specification of the used hardware.

For the models under consideration in Sections 3.1 to 3.3, we assess the algorithms’ effectiveness using the negative log-likelihood (NLL), which represents a canonical quality measure for probabilistic models (e.g., Theis et al., 2016). In experimental settings where the

em-MFA algorithm can be used as a reference, we employ the relative NLL given by

$$\text{rel. NLL} = \frac{\text{NLL}_{\text{algo}} - \text{NLL}_{\text{em-MFA}}}{\text{NLL}_{\text{em-MFA}}}, \quad (15)$$

where NLL_{algo} refers to the NLL of the respective algorithm (e.g., v-MFA), and where $\text{NLL}_{\text{em-MFA}}$ is the NLL of the em-MFA algorithm. Distance-based metrics that do not account for correlations within components (such as the quantization error used for k -means) are much less appropriate for our analysis. In the denoising benchmark in Section 3.4, we quantify denoising performance using peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM).

3.1 Scalability

The v-MFA algorithm exhibits a reduced complexity per iteration compared to em-MFA (cf. Section 2.1). However, a reduction per iteration does not guarantee a reduced complexity for the overall training. Increasingly many iterations could negate the efficiency gain per iteration. Furthermore, v-MFA requires updating the additional variational parameters.

To investigate the scalability of v-MFA and em-MFA, we measured their overall computational demands per data point as the number of mixture components C increases. For effective training, increasing model parameters commonly requires a corresponding increase in the amount and complexity of the data. Maintaining a constant data set size while increasing C could simplify optimization, typically resulting in a reduced number of iterations. To mitigate this effect, we simultaneously increase the data set size alongside C . We constructed these data sets by uniformly subsampling $\tilde{N}$ data points for each value of C , where $\tilde{N} = N \cdot C / C_{\max}$, maintaining a constant ratio of $\tilde{N}/C$. For the largest value $C = C_{\max}$, the entire data set was used. Our scalability evaluation differs in detail from previous evaluations (e.g., Hirschberger et al., 2022), but we also adjusted data set size with C to measure the efficiency, mitigating secondary effects on the number of iterations.

We applied v-MFA and em-MFA to all four benchmark data sets, with component numbers ranging from $C = 100$ to 800 in steps of 100, while fixing the hyperplane dimensionality to $H = 5$. In addition to these hyperparameters, the v-MFA algorithm includes the variational hyperparameters C' and G . While a precise definition of these variational hyperparameters is provided in Section 2, $S = C'G + 1$ defines the size of the search spaces, which control the upper limit of joint probabilities evaluated per iteration by the v-MFA (with $S \leq C$). Consequently, these variational hyperparameters balance optimization quality and computational efficiency in the algorithm.

We used the same variational hyperparameters ($C' = 3$ and $G = 15$) across all data sets and component numbers. For the effect of different hyperparameters on the results, we refer to the experiments in Section 3.2. The v-MFA and em-MFA algorithms iterate (variational) EM updates until a convergence criterion is reached (see Equation 14).

The results of the scalability analysis are summarized in Figure 4. To quantify the scalability with C , we fitted regressions of the form $f(C) = bC^a$. This corresponds to linear functions

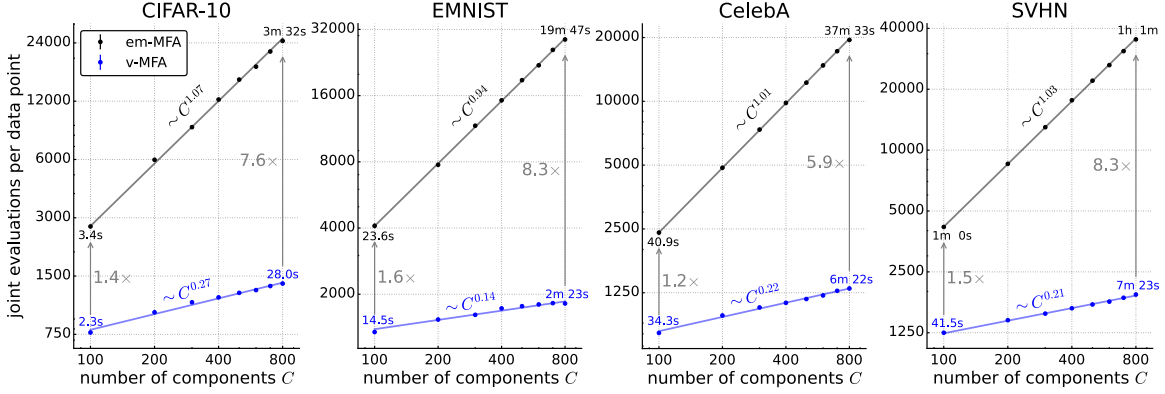


Figure 4: Scaling behavior of v-MFA and em-MFA. One measurement point shows, for a given GMM with C components, the number of joint probabilities computed per data point until the convergence criterion is reached. The number of components C is increased from $C = 100$ to $C = 800$, and subplots show the results for the data sets CIFAR-10, EMNIST, CelebA and SVHN. Each measurement point is averaged over 40 independent runs, with error bars indicating standard error of the mean (SEM). Note that the SEM values are very small and barely visible in the plot. The legend in the first subfigure applies to all four subfigures. Lines represent regressions used to determine the scaling exponent of C (see main text), while annotations provide information on total wall-clock runtimes. Additionally, gray vertical arrows indicate the relative wall-clock speed-up of v-MFA (i.e., the wall-clock runtime required by em-MFA divided by the wall-clock runtime required by v-MFA).

$\log(f(C)) = \tilde{b} + a \log(C)$ in log-log plots. Hence, the slopes in Figure 4 provide the scaling exponents for C .

For em-MFA, each iteration requires precisely NC joint evaluations, i.e., C evaluations per data point. The number of iterations tends to increase slightly with higher C , as also previously observed (e.g., Hirschberger et al., 2022). This leads to a scaling exponent slightly above one for CIFAR-10, CelebA, and SVHN, indicating slightly superlinearly scaling with C . Conversely, for EMNIST, the exponent falls slightly below one, as em-MFA (and v-MFA) required fewer iterations to converge with increasing C .

In contrast to em-MFA, the number of joint evaluations per data point and iteration is upper-bounded for v-MFA, i.e., it is limited by the search space size $S = C'G + 1 \leq C$ (cf. Section 2.3). In practice, the actual number is often lower than this bound. It may be that v-MFA requires more iterations than em-MFA due to its partial E-steps, and we indeed observe more iterations in our empirical evaluations. Importantly, however, our empirical results for v-MFA clearly show a substantial overall gain in computational efficiency compared to em-MFA. This gain presents itself as a different complexity scaling with C (in contrast to, e.g., gains in terms of constant offsets). Concretely, the joint evaluations per data point clearly scale *sublinearly* with C , with all scaling exponents consistently below $1/3$ (cubic root dependency, $\sqrt[3]{C}$).

For instance, for $C = 100$, em-MFA required approximately three times as many joint evaluations compared to v-MFA across all data sets. However, as C increased to 800, the difference spanned an order of magnitude. For even larger C , the reduction in joint evaluations for v-MFA compared to em-MFA increases still further (we investigate such larger C in the next sections).

When observing the training time (depicted as annotations in Figure 4), we discern marginal speed-ups of v-MFA over em-MFA for $C = 100$. For $C = 800$, v-MFA significantly outperformed em-MFA, which shows not only the improved theoretical complexity scaling but also practical runtime reductions. Analogous to the joint evaluations, the improvements in runtimes increase with larger values of C (cf. Section 3.3).

The results in Figure 4 so far consider solely the computational costs, not the quality of the optimization. To ensure that the observed improvements of v-MFA are not primarily due to a loss in quality, additional control experiments were conducted (see Appendix B.2.3). When we compare the computational demand required to achieve the same optimization quality (in terms of NLL_{test}), the speed-up of v-MFA over em-MFA remains essentially consistent with Figure 4. Only for EMNIST, where v-MFA showed the strongest speed-ups, a non-negligible (but small) fraction of the speed-up is due to a slight reduction in quality.

3.2 Quality vs. Efficiency Analysis

In Section 3.1, it was shown that the number of joint evaluations per data point scales sublinearly w.r.t. the number of components C for v-MFA. While those experiments focused on complexity scaling with C , we will now systematically compare optimization quality and efficiency by evaluating different algorithms on these four benchmark data sets.

We evaluated different algorithms, including em-MFA and v-MFA, by comparing wall-clock runtimes against optimization quality, measured by relative NLL values on the test sets. To provide references, we also included NLL values after initialization to highlight the learning improvements of the algorithms. Additionally, v-MFA and em-MFA were compared to the k -means+FA algorithm, which integrates k -means with factor analyzers (details in Appendix B.1.1). This approach follows Richardson and Weiss (2018), who used k -means and factor analysis for MFA initialization.

For the four data sets, we selected different numbers of components: $C = 800$ for CIFAR-10, $C = 1200$ for CelebA, $C = 1800$ for SVHN, and $C = 2000$ for EMNIST. The hyperplane dimension was set to $H = 5$ across all data sets. For v-MFA, we evaluated the performance across nine settings of the variational hyperparameters (all combinations of $C' \in \{3, 5, 7\}$ and $G \in \{5, 15, 30\}$). To initialize component means for the algorithms, we used the same initialization procedure. The same applies for the variances for v-MFA and em-MFA.

Figure 5 shows the measured NLL_{test} values vs. wall-clock runtimes for all algorithms. As can be observed, v-MFA (with $C' = 3$ and $G = 5$) was the fastest among the algorithms evaluated on all four data sets (excluding Initialization). Across all settings of hyperparameters C' and G , v-MFA consistently outperformed em-MFA, achieving substantial

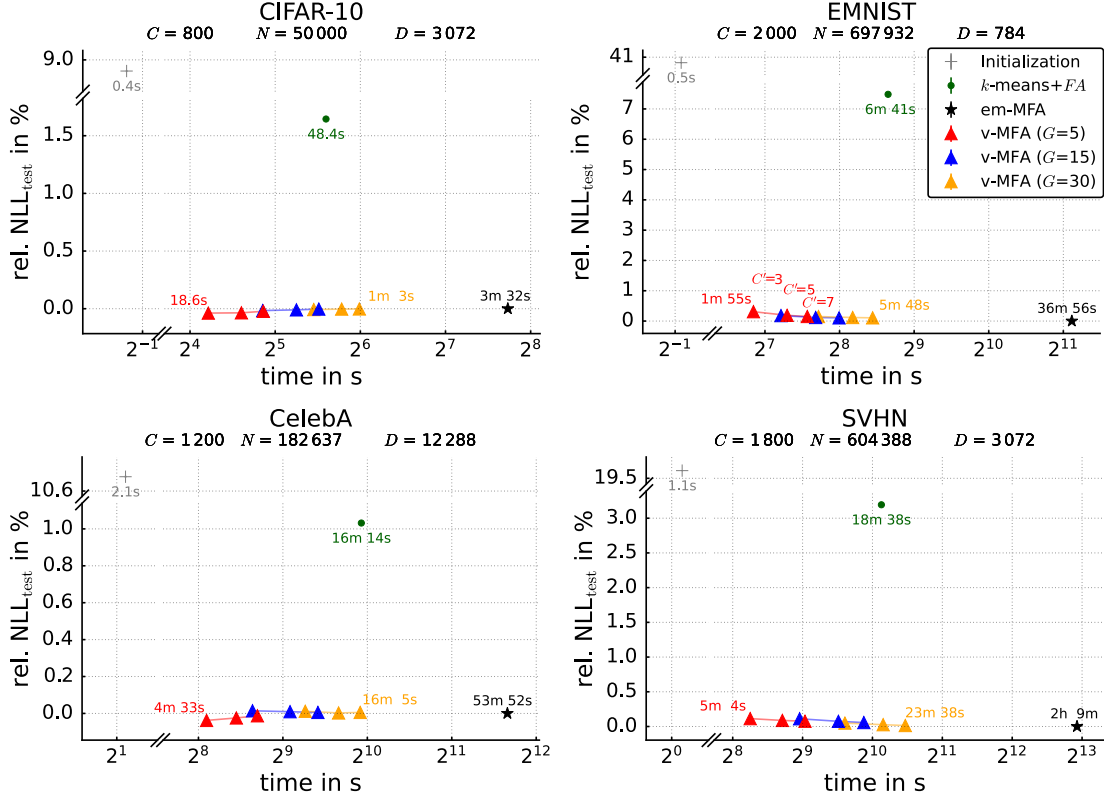


Figure 5: Effectiveness and efficiency in terms of negative log-likelihood (NLL) and runtime of v-MFA compared to em-MFA, k -means+ FA and the parameters after initialization. Each of the four subfigures refers to experiments on one benchmark data set. The y-axes denote the relative NLL on the test set w.r.t. em-MFA as baseline, given by Equation (15). For v-MFA, we present the performance across nine different settings of the hyperparameters C' and G . For each configuration of $G \in \{5, 15, 30\}$, measurements for v-MFA (three red, blue and yellow triangles, respectively) refer to configurations with $C' \in \{3, 5, 7\}$. Settings with larger C' lie to the right, as they required longer runtimes. Each measurement point is averaged over 40 independent runs, with error bars indicating standard error of the mean (SEM) (note that the SEM values are smaller than the symbol sizes).

speed-ups ranging from $3.3\times$ to $25.6\times$. Additionally, v-MFA achieved faster runtimes than k -means+ FA for the majority of hyperparameter settings.

In terms of optimization quality, em-MFA achieved the best (lowest) NLL values on the EMNIST and SVHN data sets, while v-MFA shows the best NLL values on the CIFAR-10 and CelebA data sets (with $C' = 3$ and $G = 5$). The relative deviations of v-MFA from the baseline remained consistently below 0.32%, with most settings of the hyperparameters C' and G showing smaller deviations. The k -means+ FA algorithm showed significantly higher NLL values, exhibiting a deviation from the baseline by approximately 1% for data sets

such as CelebA and CIFAR-10, and up to 3% and 7% for the SVHN and EMNIST data sets, respectively. Further details on these results are provided in Appendix B.3.

The performance of the v-MFA algorithm relative to em-MFA depends, among other factors, primarily on two aspects: (1) how well the assumption of truncated posteriors fits the data, and (2) how effectively the variational EM algorithm optimizes the variational parameters. To provide a more detailed analysis of the efficiency-quality trade-off between v-MFA and em-MFA, Appendix B.2.4 presents an ablation study. This study examines the extent to which the observed differences in optimization performance between v-MFA and em-MFA can be attributed to the use of truncation (with the optimal $\mathcal{K}_n^{\text{opt}}$) versus the variational approximation employed to determine the $\mathcal{K}_n$ sets.

Additionally, we present control experiments in Appendix B.2.2 comparing v-MFA, em-MFA and k -means+ FA with GMMs with diagonal covariances and previously studied large-scale MFA optimization (Richardson and Weiss, 2018) that used a stochastic gradient descent (SGD) approach. To evaluate the robustness and optimization quality of the v-MFA algorithm under different initialization strategies, we also conducted an ablation study, as reported in Appendix B.2.5.

3.3 GMMs with up to Billions of Parameters

While previous numerical experiments were conducted on large-scale benchmarks, they still allowed for the application of em-MFA (or k -means+ FA). The sublinear scaling of v-MFA enables addressing much larger scale problems, however. In this section, we therefore investigate significantly larger problems, which far exceed the problem sizes that can be addressed by em-MFA or other GMM approaches (on current, reasonable compute resources). Concretely, we used a data set of approximately 100M images, each sized $D = 32 \times 32 \times 3 = 3072$. The used data set is a version of the YFCC100M data set, with 5% of the samples randomly selected for testing and the remaining 95% for training (see Appendix B.1.3). We then applied v-MFA with increasingly many components. We used $C = 500, 5000, 50\,000$, and $500\,000$. Unlike in Section 3.1, optimization by v-MFA used all training samples, regardless of the chosen value of C . The other hyperparameters were set as in Section 3.1, specifically $H = 5$, $C' = 3$, and $G = 15$.

Large numbers of components have consequences for auxiliary parts of the optimization. First, our default initialization method (AFK-MC²) becomes impractical for the large C values due its quadratic computational complexity w.r.t. C . Thus, we employed a less sophisticated initialization approach. For $\mu_{1:C}$, we selected data points uniformly at random (without duplicates). The remaining parameters were initialized as before. Despite being less sophisticated, this initialization approach is expected to yield stable performance; as shown in Appendix B.2.5, the v-MFA algorithm exhibits robustness to the choice of initialization method on the data sets considered in Sections 3.1 and 3.2. Second, to further reduce initialization cost, the convergence threshold for the initialization procedure of the variational parameters, referred to as *warm-up*, was set to $\varepsilon = 10^{-3}$, a larger value than in the previous smaller-scale experiments (see Section 2.4). And third, we observed that there were components containing no data points in the cases of $C = 50\,000$ and $500\,000$.

These components were ignored, effectively reducing C , but the fraction of such ‘empty’ components was (in all experiments) less than 0.2% of all components.

Using the described data and MFA optimization, we obtain results for the computational costs and NLL values, which are shown in Figure 6. In the figure, we also express the size of an MFA representation in terms of the total number of optimized floating point parameters, which follows recent customs in studies on large-scale data representations (Dosovitskiy et al., 2021; Fang et al., 2023; Wortsman et al., 2023; Kaplan et al., 2020; Henighan et al., 2020). For an MFA representation, the number of parameters is determined by the number of components C , the hyperplane dimensionality H and the data dimensionality D . Concretely, the number of MFA parameters is given by the number of parameters for the prior, means, factor loadings and variances, i.e.,

$$C + CD + CDH + CD = C(D(H + 2) + 1).$$

In Figure 6A, we observe that both the number of joint evaluations and training runtime increased only moderately as C increased. This behavior results from the sublinear scaling of v-MFA. From $C = 50\,000$ to $C = 500\,000$, runtimes even slightly decreased, due to a small reduction in the number of iterations. At the same time, increasing the number of components C significantly reduced the test set NLL (Figure 6B). For $C = 500$, the NLL values for em-MFA and v-MFA were within the standard error of the mean, consistent with the findings in Section 3.2, where only minor NLL differences were observed between the two algorithms. Scaling em-MFA to component numbers significantly larger than $C = 500$ was computationally prohibitive. Already for $C = 500$, em-MFA required approximately two days to train compared to approximately 5 hours for v-MFA (and em-MFA required about $16\times$ as many joint evaluations). For $C = 5000$ the runtime of em-MFA would still be slower (by about an order of magnitude). Additionally, Figure 6A shows that v-MFA requires even fewer joint evaluations for the entire optimization procedure than em-MFA does for a single iteration.

Our largest scale experiments for v-MFA demonstrate that with current hardware,⁵ exceptionally large MFA models can be trained efficiently (see Figure 6). Concretely, for $C = 500\,000$ an MFA model with 10.8 billion parameters can be optimized in less than nine hours.

3.4 Benchmarking on the Downstream Task of Image Denoising

Negative log-likelihood (NLL) as used in the previous subsections is the canonical measure to evaluate the ability of probabilistic generative models to learn data densities (e.g., Theis et al., 2016). It can, however, be very instructive to add evaluations based on other measures that are obtained in specific contexts.

One strategy is to choose one of the many tasks that can be addressed by a trained generative model, e.g., denoising, deblurring, inpainting, outlier detection, compression and many more. For a probabilistic generative model, any such task is a *downstream task* in the

5. We used a node with considerable compute resources: v-MFA was trained utilizing all 64 cores of a single AMD Genoa EPYC 9554 CPU and 4 TB of RAM.

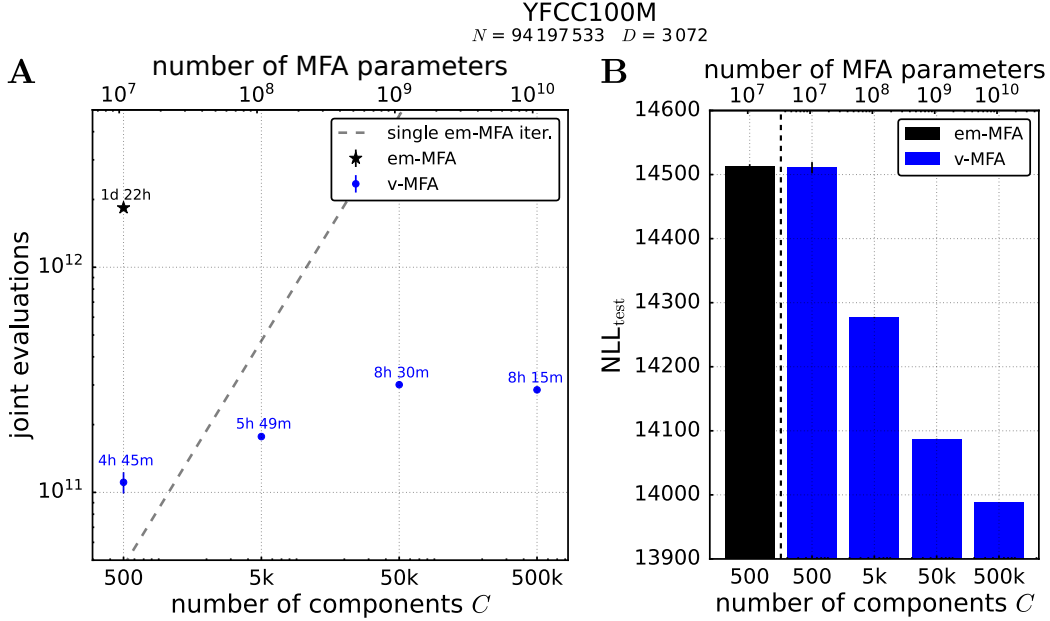


Figure 6: Evaluation of v-MFA and em-MFA on the YFCC100M data set. **A** shows for different numbers of components C the total number of joint evaluations required for one v-MFA optimization run. The number of v-MFA model parameters is shown at the top. For each C we also provide the total runtime as annotation. Additionally, the dashed gray line represents the number of joint evaluations for a *single* iteration of em-MFA (which is given by NC). **B** shows the values of the negative log-likelihood per data point on the test set for em-MFA with $C = 500$ and for v-MFA with different numbers of components C . In both figures, each measurement is averaged over 3 independent runs and error bars denote the standard error of the mean (SEM).

sense that a post-processing routine uses the learned data density in order to address the respective task. Among all possible tasks, denoising is exceptionally widely used, and the task represents a long-standing benchmark of constantly high theoretical and practical interest. Because of its importance, algorithms are often specialized only for this specific task. In contrast, any model with focus on data density learning is itself much more general-purpose. Hence, a benchmark comparing special purpose algorithms to general-purpose approaches such as generative models can be expected to favor the specialists. Nevertheless, we here choose the specific task of image denoising and a comparison to specialist algorithms developed for this task. An advantage of using image denoising as downstream benchmark is the ability to compare the effectiveness and efficiency of v-MFA to a broad range of other algorithms. Our benchmarking will thus include comparisons to approaches such as filtering (Dabov et al., 2007), self-supervised deep neural networks (DNNs; Batson and Royer, 2019a; Krull et al., 2019a; Quan et al., 2020a; Lequyer et al., 2022a), and approaches applying DNNs within probabilistic models (we will use a variational autoencoder, Prakash

et al., 2021b) or within inverse problem settings (we will use the deep image prior approach, Ulyanov et al., 2018b).

Our comparison uses the recent benchmarking protocol proposed by Lequyer et al. (2022a), which assesses a range of popular *blind zero-shot denoising* methods for images. The blind zero-shot setting has recently become very popular (e.g., Lequyer et al., 2022a; Ma et al., 2025b; Salwig et al., 2024; Mousavi and Lücke, 2025) as it avoids the training on large data sets of ‘clean’ (i.e., non-noisy) images. Such training on potentially large corpora is disadvantageous because it is often computationally costly, can introduce biases, and hence can lead to artifacts that are difficult to control (for discussions see Laine et al., 2021; Shocher et al., 2018). Furthermore, clean data is often simply not available (e.g., high resolution microscopy images, see Salwig et al., 2024, for a discussion).

Following Lequyer et al. (2022a), our evaluation data sets include two natural image data sets (BSD68 and Set12) as well as a real-world microscopy data set containing inherently noisy images, referred to as *Confocal*. BSD68 and Set12 contain grayscale natural images. Noisy versions of these images are generated by adding Gaussian noise with the standard deviations σ as listed in Table 1. The Confocal data set is a subset of confocal microscopy images from the Fluorescence Microscopy Data set (FMD; Zhang et al., 2019) and contains only images in which the noise emerges from the image recording process (i.e., the noise is not artificially added but emerges naturally). To obtain (pseudo) ground-truth references for the Confocal data set, multiple captures of the same microscopy scene were averaged (for more details, we refer to Zhang et al., 2019).

To investigate the scalability of the denoising algorithms in both performance and computational cost, we include in addition to the data used by Lequyer et al. (2022a) one high resolution image of the Div2K data set (Agustsson and Timofte, 2017). The used image (with ID ‘0010’) is a colored image of size 2040×1644 and was selected prior to any experiments. It will be denoted as ‘Tiger’. As with BSD68 and Set12, we corrupt the ‘Tiger’ image with synthetic Gaussian noise. More details on all data sets are given in Appendix B.1.3.

The original benchmark compared a range of DNNs, namely Noise2Self (N2S; Batson and Royer, 2019a), Noise2Void (N2V; Krull et al., 2019a), Self2Self (S2S; Quan et al., 2020a), and Noise2Fast (N2F; Lequyer et al., 2022a) as well as the Deep Image Prior (DIP; Ulyanov et al., 2018b) approach which uses DNNs within an inverse problem setting. Here we extend the original benchmark by additionally including the variational autoencoder (VAE)-based approach DivNoising (DivN; Prakash et al., 2021b), Pixel2Pixel (P2P; Ma et al., 2025b), and the well-known filter-based method BM3D (Dabov et al., 2007). Note that the version of DivNoising considered here and BM3D are not *blind zero-shot* because they require the value of the Gaussian noise as input parameter. Hence, both algorithms are not applied to the Confocal data set, as the noise input parameter is unknown for these images.

The above extended list of image denoising algorithms is compared to v-MFA. For image denoising with v-MFA, we apply an elementary post-processing routine that makes use of the data distribution learned by v-MFA. Concretely, v-MFA is trained on image patches as data points, and learns a corresponding data density model. The learned data density is

then used to estimate the non-noisy value of each image pixel. Details on the post-processing routine are given in Appendix A.7. As v-MFA can, as a generative model, learn from noisy data (noisy patches in this case), it is well suited for blind zero-shot denoising.

For our comparison, we employed the same default hyperparameters for a given denoising algorithm across all data sets and noise levels. Details on these settings are given in Appendix B.1.1. An exception was made for the learning rates of S2S and P2P on the Confocal data set because the default settings failed to produce meaningful results (the output images were uniformly gray).

To apply v-MFA to the benchmark data sets, we set the number of mixture components to $C = 1000$. For the other hyperparameters, we employed the values previously used in Sections 3.1 and 3.3. Concretely, we set the dimensionality of the hyperplane to $H = 5$ and the variational hyperparameters are chosen as $C' = 3$ and $G = 15$. The patch size was fixed at 12×12 across all data sets, a size observed to perform well for image denoising in previous work (cf. Drefs et al., 2022).

The zero-shot denoising algorithms are individually applied to each noisy image of a given data set, i.e., for each image, both training and inference are conducted exclusively on the image itself. Training the algorithms on the entire image set would violate the assumptions of the (blind) zero-shot denoising setting. The performance of each algorithm is evaluated using peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM) and execution time measured as wall-clock runtime. For a given data set, we compute the PSNR for each image of the data set and then report the average PSNR. We proceed in the same way for SSIM and wall-clock runtime. The denoising results are summarized in Table 1 and Figure 7. To account for the varying PSNR values of the noisy images across data sets, the PSNR gain (ΔPSNR), defined as the PSNR difference of the denoised and noisy image, is reported in Figure 7A. To normalize for differences in image size across data sets, we report processing speed in kilopixels per second (following Lequyer et al., 2022a). For an image of size $W \times H \times C$, the processing speed v is computed as

$$v = \frac{W \times H \times C \times 10^{-3}}{t},$$

where C is the number of color channels (1 for grayscale and 3 for RGB) and t is the total runtime of a given algorithm to denoise this image.

When considering the results in Table 1 and Figure 7, it may first be observed that in the setting with artificially added noise, BM3D often outperforms blind zero-shot denoising algorithms if it is provided with the true noise level. In the standard blind zero-shot setting (when the noise level is unknown), the notable observation for our study is that v-MFA almost always ranks among the top three methods in terms of PSNR and SSIM values. For this setting, the S2S approach exhibits the highest PSNR and SSIM values for all cases with artificially added Gaussian noise. For the Confocal data set, which features non-artificial noise, we observed v-MFA to obtain the highest PSNR and SSIM values.

More central to our study is, however, performance in terms of computational efficiency. Efficiency in terms of processing speed was also the main focus of the here used benchmark

Data set	σ	non-blind zero-shot			blind zero-shot						
		BM3D	DivN	N2S	N2V	DIP	S2S	N2F	P2P	v-MFA	
Set12	25	PSNR	29.99 $\pm$ 1.29	26.95 $\pm$ 1.63	27.15 $\pm$ 2.47	27.35 $\pm$ 1.55	27.16 $\pm$ 1.65	30.03 $\pm$ 1.13	29.06 $\pm$ 1.16	28.80 $\pm$ 1.40	29.15 $\pm$ 1.40
		SSIM	<u>8.52</u> $\pm$ 0.30	7.80 $\pm$ 0.52	7.37 $\pm$ 1.28	7.53 $\pm$ 0.29	7.74 $\pm$ 0.53	8.49 $\pm$ 0.32	8.22 $\pm$ 0.30	8.11 $\pm$ 0.31	8.25 $\pm$ 0.33
		Time	<u>0.81</u> $\pm$ 0.33	283.20 $\pm$ 49.93	1139.62 $\pm$ 708.82	296.50 $\pm$ 27.94	37.60 $\pm$ 1.19	3262.46 $\pm$ 2218.07	3.32 $\pm$ 2.05	16.94 $\pm$ 10.93	1.37 $\pm$ 0.80
	50	PSNR	<u>26.75</u> $\pm$ 1.27	23.81 $\pm$ 3.23	23.66 $\pm$ 3.15	24.02 $\pm$ 1.57	22.75 $\pm$ 0.91	26.54 $\pm$ 1.07	25.89 $\pm$ 1.07	25.72 $\pm$ 0.90	26.12 $\pm$ 1.19
		SSIM	<u>7.68</u> $\pm$ 0.41	6.66 $\pm$ 1.49	6.17 $\pm$ 1.49	5.86 $\pm$ 0.67	5.48 $\pm$ 0.55	7.42 $\pm$ 0.37	7.24 $\pm$ 0.42	6.50 $\pm$ 0.40	7.03 $\pm$ 0.39
		Time	<u>0.83</u> $\pm$ 0.35	197.02 $\pm$ 69.79	1140.80 $\pm$ 712.30	294.17 $\pm$ 27.80	37.91 $\pm$ 1.39	3238.05 $\pm$ 2197.54	3.36 $\pm$ 2.86	17.08 $\pm$ 10.95	1.23 $\pm$ 0.73
BSD68	25	PSNR	<u>28.61</u> $\pm$ 2.52	25.11 $\pm$ 3.25	26.72 $\pm$ 3.23	26.30 $\pm$ 2.67	25.96 $\pm$ 2.84	28.46 $\pm$ 2.85	28.11 $\pm$ 2.44	27.26 $\pm$ 2.89	27.22 $\pm$ 3.33
		SSIM	<u>8.04</u> $\pm$ 0.66	6.60 $\pm$ 1.67	7.38 $\pm$ 0.92	7.06 $\pm$ 0.70	7.13 $\pm$ 0.79	7.97 $\pm$ 0.72	7.87 $\pm$ 0.61	7.50 $\pm$ 0.59	7.54 $\pm$ 0.81
		Time	<u>0.79</u> $\pm$ 0.02	224.38 $\pm$ 94.58	1198.04 $\pm$ 2.60	920.73 $\pm$ 535.47	39.68 $\pm$ 0.48	3230.29 $\pm$ 56.55	3.67 $\pm$ 0.90	20.97 $\pm$ 0.19	1.42 $\pm$ 0.32
	50	PSNR	<u>25.67</u> $\pm$ 2.60	20.72 $\pm$ 4.89	24.49 $\pm$ 2.26	23.22 $\pm$ 2.78	22.28 $\pm$ 1.62	25.70 $\pm$ 2.48	25.31 $\pm$ 2.35	24.84 $\pm$ 2.10	25.30 $\pm$ 2.66
		SSIM	<u>6.89</u> $\pm$ 0.91	5.14 $\pm$ 2.02	6.39 $\pm$ 0.80	5.16 $\pm$ 0.77	5.16 $\pm$ 0.98	6.87 $\pm$ 0.86	6.72 $\pm$ 0.80	5.91 $\pm$ 0.51	6.51 $\pm$ 0.86
		Time	<u>0.81</u> $\pm$ 0.03	157.14 $\pm$ 96.25	1185.39 $\pm$ 2.74	935.96 $\pm$ 552.42	39.99 $\pm$ 0.83	3222.79 $\pm$ 61.80	3.36 $\pm$ 0.97	21.14 $\pm$ 0.33	1.40 $\pm$ 0.24
Confocal	-	PSNR	-	-	36.20 $\pm$ 3.88	35.70 $\pm$ 3.53	34.53 $\pm$ 3.89	36.51 $\pm$ 3.39	36.60 $\pm$ 3.84	34.02 $\pm$ 2.33	36.83 $\pm$ 3.54
		SSIM	-	-	9.26 $\pm$ 0.38	9.18 $\pm$ 0.45	8.94 $\pm$ 0.64	9.28 $\pm$ 0.38	9.33 $\pm$ 0.39	9.04 $\pm$ 0.48	9.34 $\pm$ 0.37
		Time	-	-	1964.20 $\pm$ 1.22	203.80 $\pm$ 20.56	41.93 $\pm$ 3.54	5884.49 $\pm$ 140.50	8.42 $\pm$ 0.99	30.71 $\pm$ 2.24	4.43 $\pm$ 1.63
Div2K 'Tiger'	25	PSNR	<u>32.16</u>	29.00	30.02	29.59	27.07	31.95	29.66	30.67	29.19
		SSIM	<u>8.60</u>	7.57	7.95	7.63	6.96	8.56	7.79	8.16	7.83
		Time	<u>29.46</u>	2904.16	72962.49	200.57	1652.69	89821.74	217.89	554.87	47.18
	50	PSNR	<u>29.23</u>	26.70	27.04	26.35	25.93	29.56	27.14	28.29	28.03
		SSIM	<u>7.81</u>	6.64	6.83	5.77	6.60	7.92	6.92	7.22	7.41
		Time	<u>29.28</u>	2773.43	73193.76	201.57	257.96	90042.40	217.11	502.03	38.45

Table 1: Quality and runtime of several zero-shot denoising algorithms, measured by PSNR, SSIM and average runtime per image in seconds. The reported errors denote standard deviations across images in the respective dataset. The best result within each row among the blind zero-shot algorithms are marked bold, while overall best results (including the non-blind algorithms BM3D and DivN) are underlined.

protocol of Lequyer et al. (2022a). By considering Table 1, v-MFA can be observed to be the fastest blind zero-shot method by a considerable margin. The approach is about twice as fast as N2F (requiring usually half the time), and notably over 2000 $\times$ faster than S2S (see Figure 7B). The only method faster than v-MFA is the non-blind classical filter-based BM3D approach.

In Table 1 and Figure 7, we fixed the number of components to $C = 1000$. However, the number of components C (i.e., model parameters) is expected to influence the denoising performance of v-MFA. From the high resolution ‘Tiger’ image from Div2K, we can extract about 3.3M patches, which enables us to systematically study the effect of varying C . Specifically, we evaluate $C \in \{100, 1000, 10\,000\}$, keeping all other hyperparameters fixed. The corresponding results are shown in Figure 8. From Figure 8, it is evident that larger MFA representations lead to improved denoising performance in both PSNR and SSIM. Larger models can better preserve fine structures, with edges appearing sharper, as illustrated in Figure 8A. At the same time, the runtime increases only marginally, owing to the sublinear scaling of v-MFA. For a tenfold increase in model parameters (from $C = 1000$ to $C = 10\,000$), the denoising runtime is less than twice as long.

In Appendix B.4, we provide additional denoising results, including further visual comparisons, additional noise levels as well as other GMM-based denoising algorithms. This includes a comparison of v-MFA to em-MFA, as well as a comparison in which the MFA model is replaced by a GMM with diagonal covariances. Additionally, the effect of the hyperplane dimension H on the denoising performance of v-MFA is investigated.

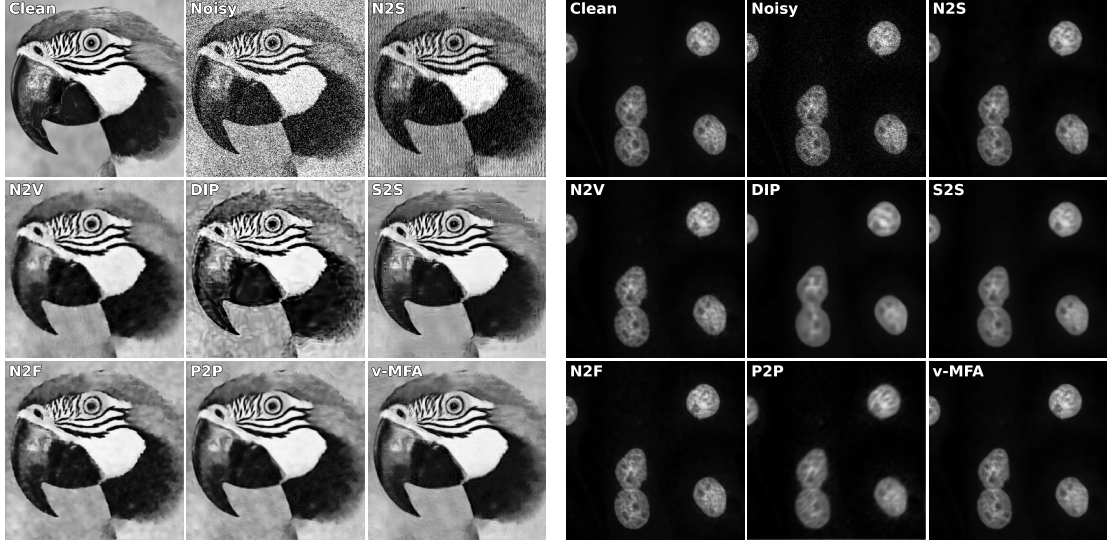
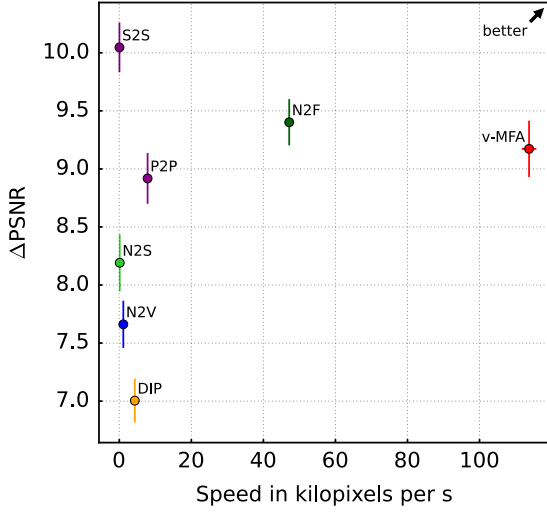
A

B


Figure 7: Denoising benchmark. **A** Results of different blind zero-shot denoising algorithms are shown for two images. On the left, denoising of the ‘parrot’ image of the Set12 data set with noise level of $\sigma = 50$ is shown. On the right, denoising results of an image section of BPAE cells from Confocal data set are shown. **B** Denoising speed in kilopixels per second and PSNR gain is shown for the blind zero-shot denoising algorithms. The plot shows averaged over all data sets and noise levels (colored dots), and error bars denote the corresponding standard error of the mean (SEM).

While em-MFA yields marginal improvements in denoising performance, it comes at the cost of very substantially increased runtimes. The results for the diagonal GMM show that the denoising performance of v-MFA is much stronger, which can be attributed to its flexible parametrization of correlations. Furthermore, the results in Table B7 show that v-MFA can achieve even higher PSNR and SSIM values than those reported in Table 1 when different hyperplane dimensions H are used. In particular, larger values of H can be beneficial when a sufficiently large number of patches is available.

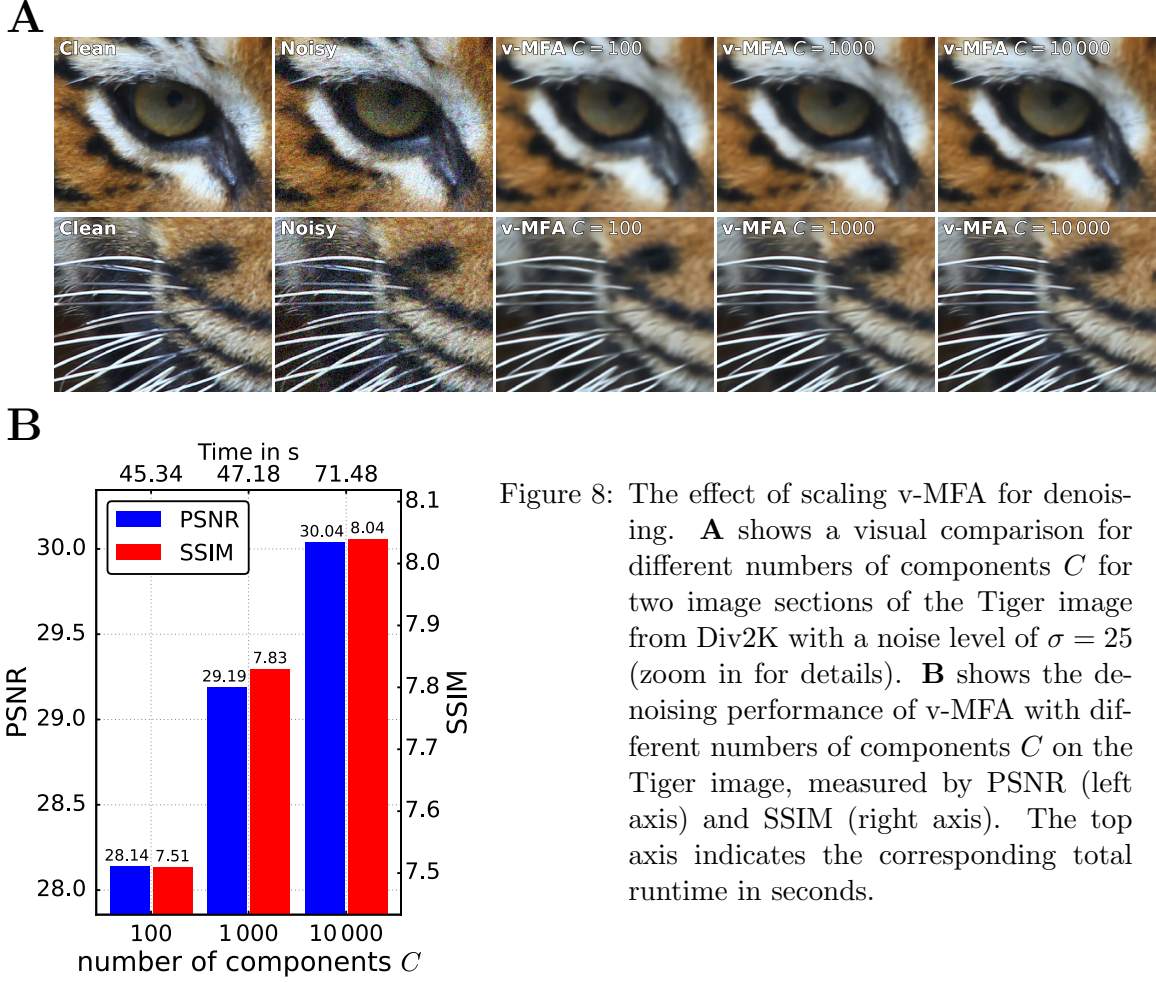


Figure 8: The effect of scaling v-MFA for denoising. **A** shows a visual comparison for different numbers of components C for two image sections of the Tiger image from Div2K with a noise level of $\sigma = 25$ (zoom in for details). **B** shows the denoising performance of v-MFA with different numbers of components C on the Tiger image, measured by PSNR (left axis) and SSIM (right axis). The top axis indicates the corresponding total runtime in seconds.

4. Discussion

One of the most salient hallmarks of GMMs is their flexible parametrization of correlations within components. Furthermore, large numbers of components C allow GMMs to approximate potentially any data density with increasing accuracy (e.g., Zeevi and Meir, 1997; Li and Barron, 1999; Maz'ya and Schmidt, 1996), and approximation quality given C decisively depends on the flexibility of GMMs. However, the flexibility of general GMMs with large C comes with a very high computational demand. For GMMs with flexible covariances we have here addressed the problem of high optimization costs. Notably, the derived and evaluated algorithm, v-MFA, does not reduce runtimes by merely a fixed constant amount or percentage. Instead, and more substantially, we have empirically shown another complexity class for runtime scaling: as the number of data points N and components C increase, the runtime complexity scales *sublinearly* with NC . Achieving such a sublinear scaling for MFA optimization was the main goal of the research here reported.

Possible future research directions are the treatment of generalizations of MFAs towards non-Gaussian observation spaces including observables of different types (Khan et al., 2010;

Banerjee et al., 2005), or extensions towards discrete time-sequence models (Rabiner, 1989; Zhou and Carin, 2015). Furthermore, future work may add other learning strategies (e.g., Zhao and Yu, 2008; Archambeau et al., 2008; Elidan and Friedman, 2002; Lin et al., 2019) to the approach presented here; and future work may explore the transferability of sublinear optimization strategies to further well-known algorithms such as different types and combinations of principal and independent component analysis (Hyvärinen, 2015; Ding and He, 2004). A crucial challenge for such future research will always be the derivation of efficient estimation of methods to define truncated distributions; and, furthermore, many models may (in contrast to GMMs and MFAs) not have closed form M-steps.

The concept of truncation, i.e., focussing posteriors by assigning probability mass only to the most relevant latent values, is likely transferable to other types of machine learning models. Notably, truncated posteriors emerge when applying the concept of mixtures of experts (Jacobs et al., 1991) to enhance the efficiency of DNNs. More concretely, gating in a standard routing approach for sparse mixtures of experts (SMoE; Shazeer et al., 2017; Fedus et al., 2022) with TopK selection is equivalent to the use of truncated posteriors in a probabilistic formulation of gating (see Nguyen-Nhat et al., 2025, and references therein for details). Efficient search procedures like developed here for v-MFA could in principle enable more efficient routing, i.e., the computation of all router scores for gating could be avoided. Especially for large numbers of experts, sublinear scaling would be a very desirable property. Furthermore, efficient modeling of correlations as used for v-MFA (and for MFA in general) could enhance gating by exploiting its probabilistic formulation (Nguyen-Nhat et al., 2025; Xu et al., 1994).

Regarding the here used models themselves, GMMs remain relatively elementary but they are also ubiquitous as the use of Gaussian is very common for standard metric data. Furthermore, non-Gaussian data typically makes the parametrization of correlations much more challenging. Like GMMs, the here investigated MFA models also still remain relatively elementary data models. Unlike general GMMs, MFAs can be regarded as additionally assuming the data to lie close to low-dimensional manifolds, which MFAs approximate piecewise by low-dimensional hyperplanes. Otherwise, MFA models do not impose any additional assumptions or structures beyond GMMs. Consequently, applications of v-MFA to tasks previously addressed using GMMs (or MFAs) should not be expected to show improvements in quality (if the same number of components is used). However, for applications involving large GMMs, the v-MFA algorithm represents a *much* more efficient optimization algorithm. At the same time, and except of the submanifold assumption, the optimized MFA models used by v-MFA retain the flexibility of general GMMs in terms of within-component correlations.

Our numerical analysis confirms that the optimization quality achieved by v-MFA is very similar to the optimization quality of conventional MFA optimization. More importantly, however, the analysis also confirms the key advantage of v-MFA. Compared to conventional training, its substantially faster optimization times were observed to result in speed-ups of an order of magnitude and more (see Sections 3.1 and 3.2). Furthermore, the relative speed-up compared to conventional training *increases* with model size. For very large scales, v-MFA is therefore the by far most efficient and likely the only computationally feasible

approach (see Section 3.3). As a consequence, v-MFA is (to the knowledge of the authors) currently the only approach allowing for the optimization of flexible GMMs with billions of parameters; and v-MFA can perform such an optimization in less than nine hours on one state-of-the-art CPU.

The application of v-MFA to image denoising as a downstream task demonstrates that scalability translates into a method that is competitive in terms of denoising performance (see Section 3.4). The observed high performance is particularly notable given that MFAs optimized by v-MFA are neither specifically designed for nor tuned to the task of image denoising. Nevertheless, v-MFA consistently outperformed several DNN-based blind zero-shot methods across nearly all evaluated settings, especially under high noise conditions. On data with naturally induced noise (the Confocal data set, Zhang et al., 2019), v-MFA even shows the highest performance. More importantly in our context, though, v-MFA achieves high denoising accuracy in substantially shorter runtimes than those observed for all compared DNN-based approaches. This makes it particularly well-suited for denoising pipelines where processing speed is critical, such as in real-time imaging applications (cf. Lequyer et al., 2022a).

Also for data sets other than image data, processing speed is often crucial. Large amounts of high dimensional data are, for instance, a typical feature of auditory data and data from genetic or proteomic sequencing (e.g., The International HapMap Consortium, 2005; Conroy et al., 2023; Bennett et al., 2023). Large-scale mixture models can be used in such context, e.g., for finding clusters, to model low-dimensional data structures or to enable downstream tasks such as error correction or missing data imputation. For such and other types of data, applications of MFA models can be based on the here reported ability of v-MFA to learn data representations at scales that were previously only accessible by another class of machine learning models, namely DNNs. For image data, the largest such neural network based models are reported to range from hundreds of millions to about one billion parameters (Dosovitskiy et al., 2021; Fang et al., 2023; Wortsman et al., 2023; also compare Hsu et al., 2021; Frey et al., 2023, for other domains). Given that GMMs are universal density approximators, the novel ability to optimize giant GMMs with billions of parameters can make them to a valuable tool for future research. This is especially relevant considering that large data sets and models are regarded as central in driving the whole field of artificial intelligence. The interpretability of GMMs and MFAs may thus allow for contributing new insights in studies on inductive biases or on how performance can scale with large model sizes. Furthermore, the here developed training principles and resulting sublinear scaling may be transferable to other (including deep) models.

Acknowledgments

We thank Jan Warnken, Roy Friedman and Yair Weiss for valuable discussions and feedback. We also thank Leonhard Kohl-Lörting for proofreading the manuscript. Furthermore, we acknowledge funding by the Lower Saxony Ministry of Research and Education (MWK Kap. 0608 TG 74), by the DFG project 471410050 (LU 1196/10-1). We also acknowledge support

by the University of Oldenburg HPC infrastructure (ROSA Compute Cluster, DFG INST 184/225-1 FUGG) for earlier project stages, and by HPC infrastructure of the University of Innsbruck (LEO/VSC/ASC).

During the preparation of this work, the authors partly used ChatGPT⁶ and DeepL⁷ in order to improve the readability and language of already written text. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Appendix A. Supplementary Information on Methods

We first provide some details about the method of truncated variational optimization. Variational distributions do, in general, define a free energy which is a tractable lower bound on the log-likelihood (see, e.g., Neal and Hinton, 1998). For a mixture model with C components, the free energy results from the log-likelihood by applying Jensen’s inequality as follows

$$\begin{aligned}\mathcal{L}(\mathbf{x}_{1:N}; \boldsymbol{\Theta}) &= \sum_{n=1}^N \log \sum_{c=1}^C p(\mathbf{x}_n, c | \boldsymbol{\Theta}) = \sum_n \log \left(\sum_c q_n(c; \tilde{\boldsymbol{\Theta}}) \frac{1}{q_n(c; \tilde{\boldsymbol{\Theta}})} p(\mathbf{x}_n, c | \boldsymbol{\Theta}) \right) \\ &\geq \sum_n \sum_c q_n(c; \tilde{\boldsymbol{\Theta}}) \log \left(\frac{1}{q_n(c; \tilde{\boldsymbol{\Theta}})} p(\mathbf{x}_n, c | \boldsymbol{\Theta}) \right) \\ &= \sum_n \left(\sum_c q_n(c; \tilde{\boldsymbol{\Theta}}) \log p(\mathbf{x}_n, c | \boldsymbol{\Theta}) + \mathcal{H}[q_n] \right),\end{aligned}$$

where the expression for the MFA model inserted is provided by Equation (5). Truncated distributions are a family of variational distributions of the form given in Equation (4), with variational parameters $\mathcal{K}$ and $\tilde{\boldsymbol{\Theta}}$. They allow for a reformulation of the free energy after each M-step that takes the form in Equation (6). Care has to be taken regarding the variational parameters $\tilde{\boldsymbol{\Theta}}$. In general, the parameters can have values different from the model parameters $\boldsymbol{\Theta}$. However, it can be shown that the optimal values of $\tilde{\boldsymbol{\Theta}}$ are equal to $\boldsymbol{\Theta}$ after each M-step. See Lücke (2019), Proposition 3, for a general derivation and see, e.g., Drefs et al. (2022) and Hirschberger et al. (2022) for expressions such as Equation (6) used for specific models. For $\tilde{\boldsymbol{\Theta}} = \boldsymbol{\Theta}$, the particularly concise expression of the free energy in Equation (6) can be derived by inserting the truncated distribution in Equation (4) into the free energy of Equation (5). As a technical remark, it can be verified (see Lücke, 2019, Proposition 1) that the above derivation of the free energy can be generalized to variational distributions $q_n(c; \tilde{\boldsymbol{\Theta}})$ with exact (‘hard’) zeros (which is a property of truncated distributions).

In the following, we provide further details on the method used to avoid inversions of large covariance matrices in Appendix A.1, the derivation of the M-step parameter updates in Appendix A.2, the proof of Proposition 1 in Appendix A.4, additional information on the estimation of component-to-component distances and the KL-divergence in Appendix A.5,

6. <https://chatgpt.com/>

7. <https://www.deepl.com/>

as well as further details on the algorithmic complexity of the variational E-step in Appendix A.6.

A.1 Avoiding Inversion of Large Covariance Matrices in MFAs

Optimizing the MFA model using EM or variational EM requires repeatedly evaluating joint probabilities $p(c, \mathbf{x}_n | \Theta)$ or, equivalently, log-joints. Thus, it is essential to assess the efficiency of log-joint evaluations. Given that

$$\begin{aligned} \log p(\mathbf{x}_n, c | \Theta) &= \log p(c | \Theta) + \log p(\mathbf{x}_n | c, \Theta) \\ &= \log \pi_c - \frac{D}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma_c| - \frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \Sigma_c^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_c), \end{aligned}$$

the crucial aspects include the evaluation of the squared Mahalanobis distance $(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \Sigma_c^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_c)$, which necessitates inverting Σ_c , and the determinant computation for $\log |\Sigma_c|$. Direct calculations involving Σ_c become computationally expensive for high-dimensional data. To enhance computational efficiency, we adopt methods from Ghahramani and Hinton (1996), McLachlan et al. (2003) and Richardson and Weiss (2018). Specifically, for the inversion of Σ_c , we utilize Woodbury’s matrix inversion lemma

$$\Sigma_c^{-1} = D_c^{-1} - D_c^{-1} \Lambda_c \left(I + \Lambda_c^\top D_c^{-1} \Lambda_c \right)^{-1} \Lambda_c^\top D_c^{-1} = D_c^{-1} - U_c V_c, \quad (\text{A1})$$

where we define

$$\begin{aligned} U_c &:= D_c^{-1} \Lambda_c \in \mathbb{R}^{D \times H}, \quad L_c := I + \Lambda_c^\top D_c^{-1} \Lambda_c = I + U_c^\top \Lambda_c \in \mathbb{R}^{H \times H}, \\ V_c &:= L_c^{-1} \Lambda_c^\top D_c^{-1} = L_c^{-1} U_c^\top \in \mathbb{R}^{H \times D}. \end{aligned}$$

Given that D_c is diagonal, its inversion is trivial. Furthermore, the matrix L_c is of size $H \times H$ and can therefore be inverted more efficiently than the $D \times D$ covariance matrix (especially for $H \ll D$). As a quantification of complexity, it is evident from Equation (A1) that the squared Mahalanobis distance, expressed as

$$\mathbf{v}^\top \Sigma_c^{-1} \mathbf{v} = \mathbf{v}^\top D_c^{-1} \mathbf{v} - (\mathbf{v}^\top U_c)(V_c \mathbf{v}),$$

can be evaluated with complexity $\mathcal{O}(DH)$ for any $\mathbf{v} \in \mathbb{R}^D$.

Finally, the log-determinant is obtained using the matrix determinant lemma

$$\log |\Sigma_c| = \log \left| \Lambda_c \Lambda_c^\top + D_c \right| = \log \left| I + \Lambda_c^\top D_c^{-1} \Lambda_c \right| + \log |D_c| = \log |L_c| + \sum_d \log \sigma_{cd}^2.$$

In conclusion, we can efficiently compute the log-joints for high-dimensional data such as images, circumventing direct computations involving covariance matrices of size $D \times D$.

The derivation for the parameter updates in the M-step that maintain efficiency for high D is provided in Appendix A.2. Although there exist alternative optimization methods to EM, such as Alternating Expectation Conditional Maximization (AECM, McLachlan et al., 2003) and Expectation Conditional Maximization (ECM, Zhao and Yu, 2008), we keep in line with the EM approach. AECM requires two cycles, involving two E-steps to update all model parameters, and the conditional M-step updates in AECM and ECM lead to higher computational complexities in D or H .

A.2 Derivation of the M-step Parameter Updates

In this section, we derive the parameter updates for the M-step of the MFA model, specified in Equation (1) in the main text, following the derivations outlined in Ghahramani and Hinton (1996). Since our parameter updates deviate from Ghahramani and Hinton (1996) in certain aspects, e.g., using a variational approach with truncated posteriors or individual diagonal variances $\mathbf{D}_c$ for each component, we present here the complete derivations for clarity and completeness.

In the following, we always denote the variational parameters $\tilde{\Theta}$ with a tilde, e.g., $\tilde{\mu}_c$, where μ_c (without a tilde) corresponds to the model parameters Θ . Recall that the MFA generative model is given by

$$p(c | \Theta) = \pi_c, \quad p(\mathbf{z} | \Theta) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad p(\mathbf{x} | c, \mathbf{z}, \Theta) = \mathcal{N}(\mathbf{x}; \Lambda_c \mathbf{z} + \mu_c, \mathbf{D}_c).$$

We start here by introducing truncated posteriors as variational distributions given by

$$\begin{aligned} q(c, \mathbf{z}; \mathbf{x}_n, \mathcal{K}_n, \tilde{\Theta}) &= \frac{1}{Z_n} p(c, \mathbf{z} | \mathbf{x}_n, \tilde{\Theta}) \delta(c \in \mathcal{K}_n) \\ &= \frac{1}{Z_n} p(c | \mathbf{x}_n, \tilde{\Theta}) p(\mathbf{z} | c, \mathbf{x}_n, \tilde{\Theta}) \delta(c \in \mathcal{K}_n) \\ &= q_n(c; \tilde{\Theta}) q_{n,c}(\mathbf{z}; \tilde{\Theta}), \end{aligned} \tag{A2}$$

and define $q_n(c; \tilde{\Theta}) := \frac{1}{Z_n} p(c | \mathbf{x}_n, \tilde{\Theta}) \delta(c \in \mathcal{K}_n)$ and $q_{n,c}(\mathbf{z}; \tilde{\Theta}) := p(\mathbf{z} | c, \mathbf{x}_n, \tilde{\Theta})$. Z_n is a normalization constant given by

$$Z_n = \sum_{\tilde{c} \in \mathcal{K}_n} \int p(\tilde{c}, \mathbf{z} | \mathbf{x}_n, \tilde{\Theta}) d\mathbf{z} = \sum_{\tilde{c} \in \mathcal{K}_n} p(\tilde{c} | \mathbf{x}_n, \tilde{\Theta}) \int p(\mathbf{z} | \tilde{c}, \mathbf{x}_n, \tilde{\Theta}) d\mathbf{z} = \sum_{\tilde{c} \in \mathcal{K}_n} p(\tilde{c} | \mathbf{x}_n, \tilde{\Theta}).$$

The distribution $q_n(c; \tilde{\Theta})$ mirrors Equation (4) in the main text, consistent with previous work employing truncated variational approximations for mixture models (e.g., Hirschberger et al., 2022; Exarchakis et al., 2022). We recognize that $q_{n,c}(\mathbf{z}; \tilde{\Theta}) = p(\mathbf{z} | c, \mathbf{x}_n, \tilde{\Theta})$ conforms to a Gaussian distribution $\mathcal{N}(\mathbf{z}; \tilde{\mu}_z, \tilde{\Sigma}_z)$, with

$$\tilde{\Sigma}_z = \left(\mathbf{I} + \tilde{\Lambda}_c^\top \tilde{\mathbf{D}}_c^{-1} \tilde{\Lambda}_c \right)^{-1} = \tilde{\mathbf{L}}_c^{-1}, \quad \tilde{\mu}_z = \tilde{\Sigma}_z \tilde{\Lambda}_c^\top \tilde{\mathbf{D}}_c^{-1} (\mathbf{x}_n - \tilde{\mu}_c) = \tilde{\mathbf{V}}_c (\mathbf{x}_n - \tilde{\mu}_c). \tag{A3}$$

In the following, we introduce the variational free energy and reformulate it into a more convenient form for deriving the parameter update equations. This free energy is given by

$$\mathcal{F}(\mathbf{x}_{1:N}; \mathcal{K}, \tilde{\Theta}, \Theta) = \sum_{n=1}^N \sum_{c=1}^C \int_{\mathbb{R}^H} q(c, \mathbf{z} | \mathbf{x}_n, \tilde{\Theta}) \log p(c, \mathbf{z}, \mathbf{x}_n | \Theta) d\mathbf{z} + \mathcal{H}[q],$$

where $\mathcal{H}[q] = -\sum_n \mathbb{E}_q[\log q(c, \mathbf{z} | \mathbf{x}_n, \tilde{\Theta})]$ denotes the Shannon entropy. Substituting Equation (A2) into the free energy yields

$$\begin{aligned} \mathcal{F}(\mathbf{x}_{1:N}; \mathcal{K}, \tilde{\Theta}, \Theta) &= \sum_{n,c} q_n(c; \tilde{\Theta}) \int q_{n,c}(\mathbf{z}; \tilde{\Theta}) \log p(c, \mathbf{z}, \mathbf{x}_n | \Theta) d\mathbf{z} + \mathcal{H}[q] \\ &= \sum_{n,c} q_n(c; \tilde{\Theta}) \mathbb{E}_{q_{n,c}}[\log p(c, \mathbf{z}, \mathbf{x}_n | \Theta)] + \mathcal{H}[q]. \end{aligned}$$

The expectation $\mathbb{E}_{q_{n,c}} [\log p(c, \mathbf{z}, \mathbf{x}_n | \boldsymbol{\Theta})]$ can be expressed as

$$\begin{aligned} \mathbb{E}_{q_{n,c}} [\log p(c, \mathbf{z}, \mathbf{x}_n | \boldsymbol{\Theta})] &= \mathbb{E}_{q_{n,c}} [\log p(c | \boldsymbol{\Theta}) + \log p(\mathbf{z} | \boldsymbol{\Theta}) + \log p(\mathbf{x}_n | c, \mathbf{z}, \boldsymbol{\Theta})] \\ &= \log \pi_c - \frac{D}{2} \log(2\pi) + \frac{1}{2} \log |\mathbf{D}_c^{-1}| - \frac{1}{2} \mathbb{E}_{q_{n,c}} \left[(\mathbf{x}_n - \boldsymbol{\mu}_c - \boldsymbol{\Lambda}_c \mathbf{z})^\top \mathbf{D}_c^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_c - \boldsymbol{\Lambda}_c \mathbf{z}) \right] \\ &= \log \pi_c - \frac{D}{2} \log(2\pi) + \frac{1}{2} \log |\mathbf{D}_c^{-1}| - \frac{1}{2} \mathbb{E}_{q_{n,c}} \left[(\mathbf{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\mathbf{z}})^\top \mathbf{D}_c^{-1} (\mathbf{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\mathbf{z}}) \right], \end{aligned} \quad (\text{A4})$$

where we combined $\boldsymbol{\Lambda}_c$ and $\boldsymbol{\mu}_c$ by defining

$$\hat{\boldsymbol{\Lambda}}_c := [\boldsymbol{\Lambda}_c \quad \boldsymbol{\mu}_c], \quad \hat{\mathbf{z}} := \begin{bmatrix} \mathbf{z} \\ 1 \end{bmatrix},$$

as we have $\hat{\boldsymbol{\Lambda}}_c \hat{\mathbf{z}} = \boldsymbol{\Lambda}_c \mathbf{z} + \boldsymbol{\mu}_c$. By making use of the symmetry of $\mathbf{D}_c^{-1}$, the cyclic property of the trace, and the linearity of $\mathbb{E}_{q_{n,c}} [\cdot]$, the expectation in Equation (A4) reformulates to

$$\begin{aligned} \mathbb{E}_{q_{n,c}} \left[(\mathbf{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\mathbf{z}})^\top \mathbf{D}_c^{-1} (\mathbf{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\mathbf{z}}) \right] &= \mathbf{x}_n^\top \mathbf{D}_c^{-1} \mathbf{x}_n - 2 \mathbf{x}_n^\top \mathbf{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}] \\ &\quad + \text{tr} \left(\hat{\boldsymbol{\Lambda}}_c^\top \mathbf{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] \right), \end{aligned}$$

Inserting the aforementioned results into the free energy yields

$$\begin{aligned} \mathcal{F}(\mathbf{x}_{1:N}; \mathcal{K}, \tilde{\boldsymbol{\Theta}}, \boldsymbol{\Theta}) &= \sum_{n,c} q_n(c; \tilde{\boldsymbol{\Theta}}) \left(\log \pi_c - \frac{D}{2} \log(2\pi) + \frac{1}{2} \log |\mathbf{D}_c^{-1}| - \frac{1}{2} \mathbf{x}_n^\top \mathbf{D}_c^{-1} \mathbf{x}_n \right. \\ &\quad \left. + \mathbf{x}_n^\top \mathbf{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}] - \frac{1}{2} \text{tr} \left(\hat{\boldsymbol{\Lambda}}_c^\top \mathbf{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] \right) \right) + \mathcal{H}[q], \end{aligned} \quad (\text{A5})$$

where the expectation values $\mathbb{E}_{q_{n,c}} [\cdot]$ are given by

$$\mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}] = \begin{bmatrix} \tilde{\boldsymbol{\mu}}_z \\ 1 \end{bmatrix}, \quad \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] = \begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_z + \tilde{\boldsymbol{\mu}}_z \tilde{\boldsymbol{\mu}}_z^\top & \tilde{\boldsymbol{\mu}}_z \\ \tilde{\boldsymbol{\mu}}_z^\top & 1 \end{bmatrix}.$$

Based on this expression of the free energy, we derive the update equations by equating the partial derivatives of Equation (A5) w.r.t. all model parameters $\boldsymbol{\Theta}$ to zero, while keeping the variational parameters $\tilde{\boldsymbol{\Theta}}$ fixed.

Updates of mixing proportions: Considering the constraint $\sum_c \pi_c = 1$, the mixing proportions are determined by the well-known expression

$$\pi_c = \frac{N_c}{N} \quad \text{with} \quad N_c = \sum_n q_n(c; \tilde{\boldsymbol{\Theta}}).$$

Updates of means and factor loadings: For the derivative w.r.t. $\hat{\boldsymbol{\Lambda}}_c$ we obtain

$$\mathbf{0} = \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{\Lambda}}_c} = \sum_n q_n(c; \tilde{\boldsymbol{\Theta}}) \mathbf{D}_c^{-1} \mathbf{x}_n \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}]^\top - \sum_n q_n(c; \tilde{\boldsymbol{\Theta}}) \mathbf{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top],$$

resulting in

$$\begin{aligned}
 \sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}]^\top &= \hat{\Lambda}_c \sum_n q_n(c; \tilde{\Theta}) \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] \\
 \Leftrightarrow \mathbf{Y}_c &= \hat{\Lambda}_c \mathbf{E}_c \\
 \Leftrightarrow \hat{\Lambda}_c &= \mathbf{Y}_c \mathbf{E}_c^{-1}.
 \end{aligned} \tag{A6}$$

where we introduced the following two matrices:

$$\mathbf{E}_c = \sum_n q_n(c; \tilde{\Theta}) \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top], \quad \mathbf{Y}_c = \sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}]^\top.$$

The factor loading matrix Λ_c is obtained from the first H columns of $\hat{\Lambda}_c$, while the mean μ_c is given by the last column of $\hat{\Lambda}_c$.

Update of variance: Taking derivatives w.r.t. $\mathbf{D}_c^{-1}$, we derive

$$\mathbf{0} = \frac{\partial \mathcal{F}}{\partial \mathbf{D}_c^{-1}} = \frac{1}{2} \sum_n q_n(c; \tilde{\Theta}) \left(\mathbf{D}_c - \mathbf{x}_n \mathbf{x}_n^\top + 2 \mathbf{x}_n \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}]^\top \hat{\Lambda}_c^\top - \hat{\Lambda}_c \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] \hat{\Lambda}_c^\top \right).$$

It follows that

$$\begin{aligned}
 \mathbf{D}_c N_c &= \sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbf{x}_n^\top - 2 \left(\sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}}]^\top \right) \hat{\Lambda}_c^\top \\
 &\quad + \hat{\Lambda}_c \left(\sum_n q_n(c; \tilde{\Theta}) \mathbb{E}_{q_{n,c}} [\hat{\mathbf{z}} \hat{\mathbf{z}}^\top] \right) \hat{\Lambda}_c^\top \\
 &= \sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbf{x}_n^\top - 2 \mathbf{Y}_c \hat{\Lambda}_c^\top + \hat{\Lambda}_c \mathbf{E}_c \hat{\Lambda}_c^\top.
 \end{aligned}$$

Substituting $\mathbf{Y}_c = \hat{\Lambda}_c \mathbf{E}_c$ into the last term, we obtain

$$\mathbf{D}_c N_c = \sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbf{x}_n^\top - \mathbf{Y}_c \hat{\Lambda}_c^\top.$$

Finally, by applying the diagonal constraint, each diagonal element σ_{cd}^2 of $\mathbf{D}_c$ can be computed as follows

$$\begin{aligned}
 \sigma_{cd}^2 &= \frac{1}{N_c} \text{diag} \left(\sum_n q_n(c; \tilde{\Theta}) \mathbf{x}_n \mathbf{x}_n^\top - \mathbf{Y}_c \hat{\Lambda}_c^\top \right)_d \\
 &= \frac{1}{N_c} \left(\sum_n q_n(c; \tilde{\Theta}) x_{nd}^2 - \sum_h \left(\mathbf{Y}_c \odot \hat{\Lambda}_c \right)_{d,h} \right),
 \end{aligned}$$

where all non-diagonal elements are set to zero and $\hat{\Lambda}_c$ are the updated parameters given by Equation (A6). Here, $\text{diag}(\cdot)$ denotes a vector constructed by the main diagonal of a given matrix and $\odot$ denotes element-wise multiplication.

Given that $q_n(c; \tilde{\Theta})$ is zero for $c \notin \mathcal{K}_n$, the computational efficiency of the above equations can be improved by summing solely over non-zero $q_n(c; \tilde{\Theta})$, as elaborated in Section 2.1 in the main text.

A.3 Relation Between Likelihood, Free Energy and Posterior Mass

Alongside contributions on improved parameter optimization, GMMs have also been studied theoretically, and notably regarding convergence guarantees. The recovery of ground truth parameters, e.g., ground truth component centers, is important for the reliability of clustering results when using GMMs. In contrast to most theoretical studies, the MFA models we here consider allow for arbitrary covariance structure along low dimensional hyperplanes. Furthermore, the v-MFA optimization considers modeling of potentially arbitrary data sets in $\mathbb{R}^D$. The setting considered here, therefore, makes it difficult to obtain rigorous theoretical results, e.g., on convergence guarantees. For more controlled settings, far-reaching theoretical insights on the optimization of GMMs and in particular on EM-based algorithms for GMMs can be obtained. For instance, works by Xu et al. (2016), Daskalakis et al. (2017) and Xu et al. (2024) even provide global convergence guarantees for EM-based GMM optimization. However, assumptions such as GMMs constrained to two components (Xu et al., 2016; Daskalakis et al., 2017) are required or strong assumptions such as Gaussianity of the data distribution (Xu et al., 2024) are used. But also results on local convergence (e.g., Kwon and Caramanis, 2020; Segol and Nadler, 2021) require significant assumptions. Essentially all approaches (for global and local convergence) assume diagonal covariance matrices per component (e.g., all those just mentioned). Furthermore, assumptions on the separation of components in the data are often central to obtain convergence guarantees (e.g., Regev and Vijayaraghavan, 2017; Diakonikolas et al., 2018; Liu and Li, 2022, and many more). Studies of both global and local convergence usually consider the properties of either conventional EM or gradient EM based optimization, which is evaluating full posteriors for all data points or batches. Variational approximations differ from conventional EM or gradient EM, and it is less clear how variational EM can be treated (even in the case when taking aside the other assumptions that are not fulfilled in our setting).

In this context, it may, however, be useful to study in some more detail the relation between the log-likelihood objective for GMMs or MFAs (see Equation 3) and the here used variational objective in Equation (5). It is, for instance, possible to exploit the specific form of the free energy objective that results from truncated variational distributions (cf. Equation 6). Using Equation (6) (also see beginning of Appendix A), we can derive the following relation for the difference between $\mathcal{L}(\mathbf{x}_{1:N}; \Theta)$ in Equation (3) and $\mathcal{F}(\mathcal{K}, \Theta)$ in Equation (5):

$$\begin{aligned} \mathcal{L}(\mathbf{x}_{1:N}; \Theta) - \mathcal{F}(\mathcal{K}, \Theta) &= \sum_n \log \left(\sum_c p(c, \mathbf{x}_n | \Theta) \right) - \sum_n \log \left(\sum_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta) \right) \\ &= \sum_n \log \left(\frac{\sum_c p(c, \mathbf{x}_n | \Theta)}{\sum_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta)} \right) = \sum_n \log \left(1 + \frac{\sum_{c \notin \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta)}{\sum_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n | \Theta)} \right). \end{aligned}$$

The ratios of the sum of joint probabilities can be rewritten as a ratio of a sum of posterior probabilities. If we furthermore use $\log(1 + x) \leq x$, we obtain

$$\mathcal{L}(\mathbf{x}_{1:N}; \Theta) - \mathcal{F}(\mathcal{K}, \Theta) = \sum_n \log \left(1 + \frac{\sum_{c \notin \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)}{\sum_{c \in \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)} \right) \leq \sum_n \frac{\sum_{c \notin \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)}{\sum_{c \in \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)}.$$

We, therefore, obtain an upper bound of $\mathcal{L}(\mathbf{x}_{1:N}; \Theta)$ in addition to the lower bound $\mathcal{F}(\mathcal{K}, \Theta)$. When additionally normalized by the number of data points N , we obtain

$$\frac{1}{N}\mathcal{F}(\mathcal{K}, \Theta) \leq \frac{1}{N}\mathcal{L}(\mathbf{x}_{1:N}; \Theta) \leq \frac{1}{N}\mathcal{F}(\mathcal{K}, \Theta) + \frac{1}{N} \sum_n \frac{\sum_{c \notin \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)}{\sum_{c \in \mathcal{K}_n} p(c | \mathbf{x}_n, \Theta)}. \quad (\text{A7})$$

The expression shows that the average of the ratio between non-captured vs. captured posterior mass directly determines how close the variational objective is to the log-likelihood. For well separated components, for instance, there is for each n one component c that will dominate the posterior values close to the global optimum. If the $\mathcal{K}_n$ sets contain the corresponding component, the average ratio will be close to zero and both bounds will be very tight. But also for any sufficiently low numbers of dominating components, tight bounds are obtained. The bounds in Equation (A7) may thus be used to directly link separability of components (or groups of components) to the relation between the variational objective and the studies (such as those discussed above) using conventional EM for the log-likelihood objective.

A.4 Proof of Proposition 1

Let $\mathcal{K}$ be a given collection of index sets $\mathcal{K}_{1:N}$. We now choose an arbitrary n and replace an index $c \in \mathcal{K}_n$ by the index $\tilde{c} \notin \mathcal{K}_n$. We denote the resulting index set by $\tilde{\mathcal{K}}_n$, and the collection of index sets $\mathcal{K}_{1:N}$ with $\mathcal{K}_n$ replaced by $\tilde{\mathcal{K}}_n$ we denote by $\tilde{\mathcal{K}}$. Now, we obtain

$$\begin{aligned} \mathcal{F}(\tilde{\mathcal{K}}, \Theta) &> \mathcal{F}(\mathcal{K}, \Theta) \\ \Leftrightarrow \sum_{\substack{n'=1 \\ n' \neq n}}^N \log \left(\sum_{c' \in \mathcal{K}_{n'}} p(c', \mathbf{x}_{n'} | \Theta) \right) + \log \left(\sum_{c' \in \tilde{\mathcal{K}}_n} p(c', \mathbf{x}_n | \Theta) \right) \\ &> \sum_{\substack{n'=1 \\ n' \neq n}}^N \log \left(\sum_{c' \in \mathcal{K}_{n'}} p(c', \mathbf{x}_{n'} | \Theta) \right) + \log \left(\sum_{c' \in \mathcal{K}_n} p(c', \mathbf{x}_n | \Theta) \right) \\ \Leftrightarrow \log \left(\sum_{c' \in \tilde{\mathcal{K}}_n} p(c', \mathbf{x}_n | \Theta) \right) &> \log \left(\sum_{c' \in \mathcal{K}_n} p(c', \mathbf{x}_n | \Theta) \right) \\ \Leftrightarrow \sum_{c' \in \tilde{\mathcal{K}}_n \setminus \{\tilde{c}\}} p(c', \mathbf{x}_n | \Theta) + p(\tilde{c}, \mathbf{x}_n | \Theta) &> \sum_{c' \in \mathcal{K}_n \setminus \{c\}} p(c', \mathbf{x}_n | \Theta) + p(c, \mathbf{x}_n | \Theta) \\ \Leftrightarrow p(\tilde{c}, \mathbf{x}_n | \Theta) &> p(c, \mathbf{x}_n | \Theta), \end{aligned}$$

where we have used strict concavity of the logarithm and $\tilde{\mathcal{K}}_n \setminus \{\tilde{c}\} = \mathcal{K}_n \setminus \{c\}$.

With the same argumentation, the free energy decreases if $p(\tilde{c}, \mathbf{x}_n | \Theta) < p(c, \mathbf{x}_n | \Theta)$. For $p(\tilde{c}, \mathbf{x}_n | \Theta) = p(c, \mathbf{x}_n | \Theta)$ the free energy trivially remains unchanged. ■

A.5 Estimation of Component-to-Component Distances and KL-Divergence

In this section, we further elaborate on the derivation of the KL-divergence approximation $D_{c\tilde{c}}$, which we use to find components $\tilde{c}$ similar to component c (Equation 12 in the main text). Afterwards, we compare $D_{c\tilde{c}}$ to a previously suggested distance estimate in Appendix A.5.1.

We start by considering idealized assumptions to derive $D_{c\tilde{c}}$ as an approximation of the KL-divergence. Importantly, however, $D_{c\tilde{c}}$ remains a valid expression for the estimation of similarity of components $\tilde{c}$ to c also if the idealized assumptions are not fulfilled (which we will elaborate on further below). The numerical experiments in Section 3 in the main text verify the validity of $D_{c\tilde{c}}$ through the effective and efficient performance of the v-MFA algorithm.

Let us first assume that v-MFA has already converged to a large extent, i.e., we assume that the GMM parameters have already reached values that accurately represent the data components, and that the search spaces $\mathcal{S}_n$ include the most likely components for their data points $\mathbf{x}_n$. Furthermore, we assume that the data is appropriately modeled by a GMM (i.e., approximately Gaussian components), and that the components are well separated. Under these idealized conditions, we can assume that $D_{c\tilde{c}}$ given by Equation (12) in the main text can approximate the KL-divergence relatively well for any components $\tilde{c}$ similar to c .

To illustrate this, we first consider the finite sample approximation of the KL-divergence (Equation 10 in the main text) using M samples of $p(\mathbf{x} | c, \Theta)$, which becomes exact for $M \rightarrow \infty$:

$$D_{\text{KL}} [p(\mathbf{x} | c, \Theta) \parallel p(\mathbf{x} | \tilde{c}, \Theta)] \approx \frac{1}{M} \sum_{m=1}^M \log \frac{p(\mathbf{x}_m | c, \Theta)}{p(\mathbf{x}_m | \tilde{c}, \Theta)}, \quad \text{where } \mathbf{x}_m \sim p(\mathbf{x} | c, \Theta). \quad (\text{A8})$$

Secondly, for well separated components, model parameters close to convergence and data that is well-modeled by Gaussian components, the partitions $\mathcal{I}_c$ of Equation (11) in the main text will contain data points close to those that would be generated by $p(\mathbf{x} | c, \Theta)$. Therefore, we can further approximate Equation (A8) by

$$D_{\text{KL}} [p(\mathbf{x} | c, \Theta) \parallel p(\mathbf{x} | \tilde{c}, \Theta)] \approx \frac{1}{|\mathcal{I}_c|} \sum_{n \in \mathcal{I}_c} \log \frac{p(\mathbf{x}_n | c, \Theta)}{p(\mathbf{x}_n | \tilde{c}, \Theta)}.$$

Finally, we introduce the condition $\delta(\tilde{c} \in \mathcal{S}_n)$, which ensures that only components $\tilde{c}$ within the search spaces $\mathcal{S}_n$ are considered. The condition is important for computational efficiency, as it avoids additional computations of $p(\mathbf{x}_n | \tilde{c}, \Theta)$ that are not performed during the partial E-step. A disadvantage is that the approximation quality may decrease as potentially fewer data points $\mathbf{x}_n$ are considered for the approximation. However, if the algorithm has sufficiently converged, the components $\tilde{c}$ in the search spaces $\mathcal{S}_n$ of all data points with $n \in \mathcal{I}_c$ will be almost the same. In that case, $\delta(\tilde{c} \in \mathcal{S}_n)$ will nearly always be equal to one,

and we obtain

$$D_{\text{KL}} [p(\mathbf{x} | c, \Theta) \parallel p(\mathbf{x} | \tilde{c}, \Theta)] \approx \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log \frac{p(\mathbf{x}_n | c, \Theta)}{p(\mathbf{x}_n | \tilde{c}, \Theta)} \delta(\tilde{c} \in \mathcal{S}_n) = D_{c\tilde{c}}, \quad (\text{A9})$$

$$\text{where } N_{c\tilde{c}} = \sum_{n \in \mathcal{I}_c} \delta(\tilde{c} \in \mathcal{S}_n) \approx \sum_{n \in \mathcal{I}_c} 1 = |\mathcal{I}_c|.$$

For sufficiently similar components $\tilde{c}$ in the sense of KL-divergences, the estimate $D_{c\tilde{c}}$ can therefore provide an accurate approximation of the KL-divergence. However, as the components $\tilde{c}$ become increasingly dissimilar, the uncertainty regarding the accuracy of this approximation increases.

On the other hand, the estimate $D_{c\tilde{c}}$ can not be expected to approximate the KL-divergence accurately if the components are strongly overlapping. In such cases, the sets $\mathcal{I}_c$ may not represent their respective components well, as each data point can only be assigned to a single component. However, $D_{c\tilde{c}}$ will by construction still result in small values as long as model parameters and search spaces $\mathcal{S}_n$ have sufficiently converged. In an extreme case of $p(\mathbf{x}_n | \tilde{c}, \Theta)$ being almost equal to $p(\mathbf{x}_n | c, \Theta)$, the summands in Equation (A9) will be close to zero. As a consequence, similar components in the KL-divergence sense will also be similar in a ranking based on $D_{c\tilde{c}}$, although values of $D_{c\tilde{c}}$ may differ from the values of the KL-divergences.

Finally, for components $\tilde{c}$ that are *very* irrelevant to c , we set the values of $D_{c\tilde{c}}$ to infinity, which would also not match the values of the KL-divergence. However, irrelevant components in the KL-divergence sense are anyway disregarded for the definition of sets g_c in Equation (13) in the main text.

To summarize, $D_{c\tilde{c}}$ serves as an approximation for the KL-divergence between $p(\mathbf{x}_n | c, \Theta)$ and $p(\mathbf{x}_n | \tilde{c}, \Theta)$, which can be quite coarse. However, as the values of $D_{c\tilde{c}}$ are only used for ranking the similarity of components, coarse estimates are sufficient. Most importantly, this approximation avoids the $\mathcal{O}(C^2 DH)$ scaling of computing the exact KL-divergences of all pairwise components. Consequently, it preserves the overall computational complexity of the variational EM algorithm, which is critical for ensuring scalability w.r.t. the number of components (cf. Algorithm 3).

A.5.1 COMPARISON TO VARIATIONAL E-STEPS OF PREVIOUS GMM OPTIMIZATIONS

Previous work using variational E-steps (Hirschberger et al., 2022; Exarchakis et al., 2022) has not considered GMMs with intra-component correlations. Estimation of component-to-component distances has, however, been used before by Hirschberger et al. (2022), who applied the estimate

$$d_{c\tilde{c}}^2 := -\frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log p(\tilde{c}, \mathbf{x}_n | \Theta) \delta(\tilde{c} \in \mathcal{S}_n), \quad (\text{A10})$$

where $N_{c\tilde{c}}$ is defined as in Equation (12) in the main text. The estimate $d_{c\tilde{c}}^2$ was then used as a ranking to define sets g_c analogously to Equation (13) in the main text.

For isotropic components with equal variances and equal mixing proportions per component, it was argued that $d_{c\tilde{c}}^2$ will ultimately correspond to the Euclidean component-to-component distance (and $d_{c\tilde{c}}^2$ was originally defined for this isotropic case). More concretely, it was shown (Appendix E of Hirschberger et al., 2022) that $d_{c\tilde{c}}^2$ results in approximately the same distance ranking of component pairs $(c, \tilde{c})$ as the Euclidean component-to-component distance $\|\boldsymbol{\mu}_c - \boldsymbol{\mu}_{\tilde{c}}\|$. For diagonal covariance matrices, $d_{c\tilde{c}}^2$ was also related to KL-divergences.

From the perspective of the here used approach in Equation (12) in the main text, however, Equation (A10) can be considered as ignoring $p(\mathbf{x}_n | c, \boldsymbol{\Theta})$ in the KL-divergence (Equation 10 in the main text). A further difference is a different consideration of the mixing proportions. Estimate Equation (A10) has been shown to perform well for isotropic and diagonal components (Hirschberger et al., 2022), for which it was originally derived. Here, our estimate is directly derived from KL-divergences, which implicitly accounts for different intra-component correlations. Consequently, the more general derivation of $D_{c\tilde{c}}$ (used for v-MFA) is a better match to these intra-component correlations, and hence can be expected to result in a more efficient learning algorithm. If and how much the more general estimate Equation (12) in the main text is improving learning remains to be numerically investigated, which we provide in Appendix B.2.1.

A.6 Algorithmic Complexity of the Variational E-step

To determine the algorithmic complexity of Algorithm 1 in the main text, we analyze each of its four blocks, assuming $N > C$. Algorithm 3 outlines the steps in Algorithm 1 in the main text along with the complexity of each block (emphasized in bold) and the intermediate steps.

Block 1: This block begins with constructing the search space, where each search space can contain up to $S = C'G + 1$ elements. Following this, the joints are computed, where each joint computation has a complexity of $\mathcal{O}(DH)$ (see Appendix A.1). Next, selecting the C' largest values among the computed joints to obtain $\mathcal{K}_n$ for a data point can be performed in $\mathcal{O}(S)$ (Blum et al., 1973). Since we need to evaluate up to S joints per data point, the overall complexity of block 1 is $\mathcal{O}(NSDH)$.

Block 2: In block 2, the data set is partitioned into $\mathcal{I}_{1:C}$. To find the most likely component c_n within the $\mathcal{K}_n$ set of each data point, the component with the largest joint is selected, which has a complexity of $\mathcal{O}(C')$. Repeating this for all data points results in a total complexity of $\mathcal{O}(NC')$.

Block 3: By construction, the union of all $\mathcal{I}_c$ contains all indices of the N data points exactly once. Therefore, in block 3, we loop over each search space of the data points exactly once, resulting in a complexity of $\mathcal{O}(NS)$. On average, $\mathcal{I}_c$ contains N/C indices. Thus, the average cost to iterate over all n in $\mathcal{I}_c$ and the respective $\mathcal{S}_n$ is $\mathcal{O}^\dagger((N/C)S)$. Similar to the update of $\mathcal{K}_n$ in block 1, the complexity of finding G replacement candidates to obtain g_c is $\mathcal{O}^\dagger((N/C)S)$ on average.

Algorithm 3: Complexity of the Variational E-step

for $n = 1 : N$ do		$\mathcal{O}(NSDH)$
	$\mathcal{S}_n = \bigcup_{c \in \mathcal{K}_n} g_c;$	$\mathcal{O}(S)$
	$\mathcal{S}_n = \mathcal{S}_n \cup \{c\}$ with $c \sim \mathcal{U}\{1, C\};$	$\mathcal{O}(1)$
1	for $c \in \mathcal{S}_n$ do	$\mathcal{O}(SDH)$
	$\perp$ compute joint $p(c, \mathbf{x}_n \Theta);$	$\mathcal{O}(DH)$
	$\mathcal{K}_n = \{c \mid p(c, \mathbf{x}_n \Theta) \text{ is among the } C' \text{ largest joints for all } c \in \mathcal{S}_n\};$	$\mathcal{O}(S)$
for $n = 1 : N$ do		$\mathcal{O}(NC')$
	$c_n = \operatorname{argmax}_{c \in \mathcal{K}_n} p(c, \mathbf{x}_n \Theta);$	$\mathcal{O}(C')$
2	$\mathcal{I}_{c_n} = \mathcal{I}_{c_n} \cup \{n\};$	$\mathcal{O}(1)$
for $c = 1 : C$ do		$\mathcal{O}(NS)$
	for $n \in \mathcal{I}_c$ do	$\mathcal{O}^\dagger((N/C)S)$
	for $\tilde{c} \in \mathcal{S}_n \setminus \{c\}$ do	$\mathcal{O}(S)$
	$\tilde{D}_{c\tilde{c}} = \tilde{D}_{c\tilde{c}} + \log p(\mathbf{x}_n c, \Theta) - \log p(\mathbf{x}_n \tilde{c}, \Theta);$	$\mathcal{O}(1)$
	$N_{c\tilde{c}} = N_{c\tilde{c}} + 1;$	$\mathcal{O}(1)$
3	$\mathcal{D}_c = \mathcal{D}_c \cup \{\tilde{c}\};$	$\mathcal{O}(1)$
	for $\tilde{c} \in \mathcal{D}_c$ do	$\mathcal{O}^\dagger((N/C)S)$
	$D_{c\tilde{c}} = \tilde{D}_{c\tilde{c}}/N_{c\tilde{c}};$	$\mathcal{O}(1)$
	$g_c = \{\tilde{c} \mid D_{c\tilde{c}} \text{ is among the } G - 1 \text{ smallest values for } \tilde{c} \in \mathcal{D}_c\} \cup \{c\};$	$\mathcal{O}^\dagger((N/C)S)$
for $n = 1 : N$ do		$\mathcal{O}(NC')$
	for $c \in \mathcal{K}_n$ do	$\mathcal{O}(C')$
	$Z_n = Z_n + p(c, \mathbf{x}_n \Theta);$	$\mathcal{O}(1)$
4	for $c \in \mathcal{K}_n$ do	$\mathcal{O}(C')$
	$q_n(c; \Theta) = p(c, \mathbf{x}_n \Theta)/Z_n;$	$\mathcal{O}(1)$

$\mathcal{O}^\dagger$ denotes amortized or average complexity

Block 4: Normalizing the variational distributions requires calculating the normalization constant Z_n by summing over all C' indices in $\mathcal{K}_n$, followed by dividing by Z_n . Consequently, the total computational complexity for all data points is $\mathcal{O}(NC')$.

Summarizing the complexities of all four blocks, the overall algorithmic complexity of Algorithm 3 is $\mathcal{O}(NSDH + NC' + NS + NC') = \mathcal{O}(NSDH)$.

Algorithm 3 is analogous to the partial variational E-step described in Hirschberger et al. (2022), with three differences. We will discuss these differences below, focusing on the complexity of the respective algorithms. First, we refrain from using coresets. In Hirschberger et al. (2022), the use of coresets reduces the complexity from N to the coreset size N' , where $N > N'$. But coresets are not desirable in our setting, as discussed in Section 1 in the main text. Second, computing a joint defined by the MFA model requires $\mathcal{O}(DH)$ operations, in

contrast to the $\mathcal{O}(D)$ operations required for the GMMs considered in Hirschberger et al. (2022). The additional cost allows for a much more flexible modeling of correlations (i.e., of component shapes) than GMMs restricted to uncorrelated data per component. Nevertheless, a quadratic scaling with D (as would be required for full covariance matrices) is avoided. Third, the update of variational parameters via the estimation of component similarity is different (see Section 2.3 in the main text, Appendix A.5.1 and Appendix B.2.1).

A.7 Postprocessing Routine for Denoising

As part of our numerical experiments, we apply v-MFA and other variants of GMMs for the task of image denoising, i.e., we aim to remove noise from an image corrupted by artificial (Gaussian) noise or non-artificial, ‘inherent’ noise (see Section 3.4 in the main text and Appendix B.4). We follow the image denoising pipeline for generative models used in Drefs et al. (2022) (also compare Salwig et al., 2024). We start by outlining the general procedure before providing details specific for the generative models used here.

For a given image, we first extract all possible overlapping patches of a specified size. Consider an image of size $W \times H (\times C)$ and a patch size of $P_W \times P_H (\times C)$, where C denotes the number of color channels ($C = 3$ for RGB images, $C = 1$ for grayscale). Using a sliding window, we obtain $N = (W - P_W + 1) \cdot (H - P_H + 1)$ noisy patches. The extracted N noisy patches are the data points. Each patch is taken to be a flattened vector $\mathbf{x}$ of length $D = W \times H (\times C)$. Note that the flattening can be done row-wise, column-wise, or in any consistent order as the used generative models (GMMs and MFAs) do not assume any structure (such as two-dimensional patch structures). Still we will refer to $\mathbf{x}$ as an image patch, as the vector contains the information of one patch.

For each noisy patch $\mathbf{x}$, our goal is to estimate a corresponding ‘clean’ patch $\mathbf{x}^{\text{est}}$. Therefore, the data model is trained on the noisy patch set $\mathbf{x}_{1:N}$ (see, e.g., Section 2.4 and Algorithm 2). After inferring the model parameters, the learned representation is used to estimate the non-noisy patches. Here we use a probabilistic data estimator (compare Drefs et al., 2022), as described in detail below. With clean versions of each extracted patch, the image is reconstructed by merging these patches. As overlapping patches are used, each pixel in the image has multiple estimates. The single final pixel value is determined by computing the median of all corresponding estimates for a given pixel. A schematic overview of this denoising pipeline for generative models is shown in Figure A1.

In the following, we derive the probabilistic data estimator for MFAs and GMMs with isotropic or diagonal covariance matrices.

MFAs: For the MFA model, the denoised pixel values of a given image patch $\mathbf{x}$ are obtained by

$$\begin{aligned} \mathbf{x}^{\text{est}} &= \mathbb{E}_{p(c, \mathbf{z} | \mathbf{x}, \Theta)} [\Lambda_c \mathbf{z} + \boldsymbol{\mu}_c] \\ &= \sum_c \int p(c, \mathbf{z} | \mathbf{x}, \Theta) (\Lambda_c \mathbf{z} + \boldsymbol{\mu}_c) d\mathbf{z}. \end{aligned} \tag{A11}$$

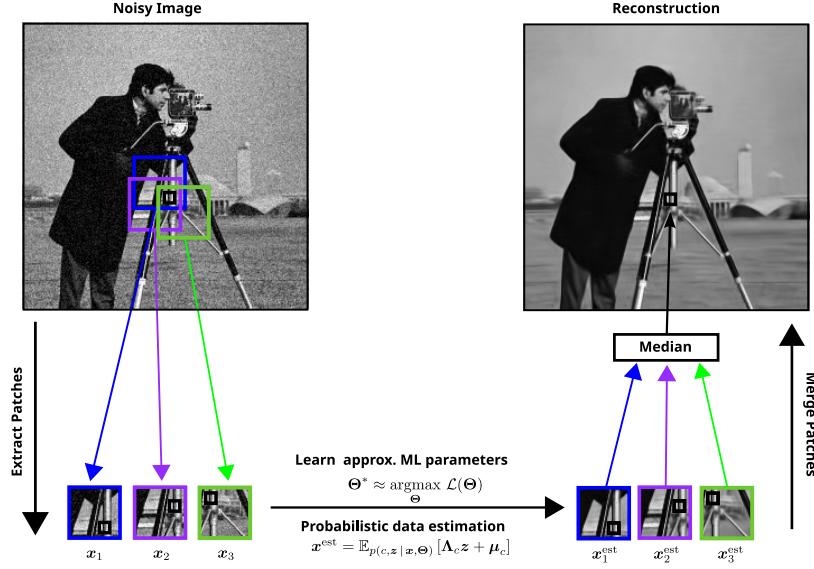


Figure A1: Illustration of patch-based blind zero-shot image denoising using probabilistic generative models. A noisy image serves as input (top left). Patches extracted from the noisy image (bottom left) are used to learn a probabilistic representation of the underlying image structure via an appropriate generative model (e.g., MFAs). Non-noisy patches are then estimated using this learned representation (bottom right). The final denoised image is obtained by aggregating pixel estimates from overlapping patches, using the median (top center and right). This approach requires no clean training data, and can be directly applied to a single noisy image.

The joint posterior distribution $p(c, \mathbf{z} \mid \mathbf{x}, \Theta)$ can be factorized as

$$p(c, \mathbf{z} \mid \mathbf{x}, \Theta) = p(c \mid \mathbf{x}, \Theta) p(\mathbf{z} \mid c, \mathbf{x}, \Theta). \quad (\text{A12})$$

Inserting this in Equation (A11) yields

$$\begin{aligned} \mathbf{x}^{\text{est}} &= \sum_c \int p(c \mid \mathbf{x}, \Theta) p(\mathbf{z} \mid c, \mathbf{x}, \Theta) (\Lambda_c \mathbf{z} + \mu_c) d\mathbf{z} \\ &= \sum_c p(c \mid \mathbf{x}, \Theta) \Lambda_c \int p(\mathbf{z} \mid c, \mathbf{x}, \Theta) \mathbf{z} d\mathbf{z} + \sum_c p(c \mid \mathbf{x}, \Theta) \mu_c \\ &= \sum_c p(c \mid \mathbf{x}, \Theta) \Lambda_c \mathbb{E}_{p(\mathbf{z} \mid c, \mathbf{x}, \Theta)} [\mathbf{z}] + \sum_c p(c \mid \mathbf{x}, \Theta) \mu_c. \end{aligned} \quad (\text{A13})$$

The posterior $p(\mathbf{z} \mid c, \mathbf{x}, \Theta)$ conforms to a Gaussian distribution $\mathcal{N}(\mathbf{z}; \mu_z, \Sigma_z)$ with $\Sigma_z = \mathbf{L}_c^{-1}$ and $\mu_z = \mathbf{V}_c(\mathbf{x} - \mu_c)$ (compare Equation A3). Consequently, it holds that $\mathbb{E}_{p(\mathbf{z} \mid c, \mathbf{x}, \Theta)} [\mathbf{z}] = \mu_z$.

Inserting this into Equation (A13) yields

$$\begin{aligned}
\mathbf{x}^{\text{est}} &= \sum_c p(c | \mathbf{x}, \Theta) \Lambda_c \boldsymbol{\mu}_c + \sum_c p(c | \mathbf{x}, \Theta) \boldsymbol{\mu}_c \\
&= \sum_c p(c | \mathbf{x}, \Theta) \Lambda_c \mathbf{V}_c (\mathbf{x} - \boldsymbol{\mu}_c) + \sum_c p(c | \mathbf{x}, \Theta) \boldsymbol{\mu}_c \\
&= \sum_c p(c | \mathbf{x}, \Theta) (\Lambda_c \mathbf{V}_c (\mathbf{x} - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c) \\
&= \mathbb{E}_{p(c | \mathbf{x}, \Theta)} [\Lambda_c \mathbf{V}_c (\mathbf{x} - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c].
\end{aligned} \tag{A14}$$

For an MFA model trained with em-MFA, Equation (A14) is used to reconstruct patches. However, for an MFA model trained with v-MFA, the expectation can then be efficiently approximated by replacing the exact posteriors $p(c | \mathbf{x}_n, \Theta)$ with the truncated posteriors $q_n(c; \Theta)$ defined in Equation (4), as follows

$$\mathbf{x}^{\text{est}} \approx \mathbb{E}_{q_n(c; \Theta)} [\Lambda_c \mathbf{V}_c (\mathbf{x}_n - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c]. \tag{A15}$$

Isotropic and diagonal GMMs: For GMMs with isotropic or diagonal covariance matrices, the denoised pixel values of a given image patch $\mathbf{x}_n$ are obtained by

$$\mathbf{x}^{\text{est}} = \mathbb{E}_{p(c | \mathbf{x}_n, \Theta)} [\boldsymbol{\mu}_c]. \tag{A16}$$

For a GMM model trained with our variational approach, we efficiently approximate the posterior $p(c | \mathbf{x}_n, \Theta)$ with the variational posterior $q_n(c; \Theta)$:

$$\mathbf{x}^{\text{est}} \approx \mathbb{E}_{q_n(c; \Theta)} [\boldsymbol{\mu}_c]. \tag{A17}$$

Appendix B. Supplementary Information on Numerical Experiments and Control Experiments

In the following, we provide further specifics on the implementation and execution of the algorithms, the used hardware and the data sets in Appendix B.1, results of additional control experiments in Appendix B.2 and supplementary information on the quality analysis in Appendix B.3.

B.1 Further Details on Numerical Experiments

Below, we present additional details regarding the implementation and execution of the algorithms as well as information about the used hardware and the data sets.

B.1.1 ALGORITHMS

The em-MFA and the v-MFA algorithms are our own implementations. To ensure a fair comparison, the primary motivation behind developing these algorithms was to optimize for execution speed and efficient memory usage. The *torch-mfa* algorithm is a published PyTorch-based implementation of MFA that uses Stochastic Gradient Descent (SGD). The *k-means+FA* algorithm combines *k*-means with factor analysis. While it was used as an

initialization method in Richardson and Weiss (2018), we employ it here as a comparison method to the other algorithms.

v-MFA: The v-MFA algorithm trains the MFA model using the variational EM algorithm described in Sections 2.1 to 2.4 in the main text. The implementation uses both Python and C++. The core functionality is written in C++, utilizing the open-source libraries Eigen (Guennebaud et al., 2010) and Boost (The Boost organization, 2024). To facilitate data communication between Python and C++, pybind11 (Jakob et al., 2017) is used. The v-MFA source code is available on GitHub.⁸ The implementations of v-MFA^{Eucl.}, v-ISO and v-ISO^{Eucl.} used in Appendix B.2 as well as v-DIAG used in Appendices B.2.2 and B.4 are also provided in this source code. The source code of the denoising extension for the v-MFA algorithm is also available on GitHub.⁹

em-MFA: Our Python implementation of the em-MFA algorithm optimizes the MFA model using exact inference via conventional EM (full posteriors). It primarily leverages the open-source library PyTorch (Ansel et al., 2024). The em-MFA source code is available on GitHub.¹⁰ The implementation em-DIAG used in Appendices B.2.2 and B.4 is also provided in this source code. The source code of the denoising extension for the em-MFA algorithm is also available on GitHub.¹¹

k-means+FA: In this integrated methodology, the initial step entails the application of the k -means algorithm to the data set. Upon convergence, the data is partitioned into Voronoi cells, with each data point $\mathbf{x}_n$ assigned to its closest cluster (component) c , i.e.,

$$\mathcal{P}_c = \{n \mid c = c_n\} \quad \text{with} \quad c_n = \underset{c'}{\operatorname{argmin}} \|\mathbf{x}_n - \boldsymbol{\mu}_{c'}\|^2. \quad (\text{B1})$$

A factor analyzer is then applied to each data subset $\mathcal{P}_c$, where each factor analyzer independently learns the parameters $\boldsymbol{\mu}_c$, $\boldsymbol{\Lambda}_c$ and $\mathbf{D}_c$ for the corresponding cluster c . The priors π_c are determined based on the relative subset sizes, i.e.,

$$\pi_c = \frac{|\mathcal{P}_c|}{N},$$

where N is the number of the total data points. The k -means+FA algorithm is realized by using the Scikit-learn (Pedregosa et al., 2011) implementations (version 1.5.0) of k -means and Factor Analyzer, using their default hyperparameters and settings.

torch-mfa: We used the official *torch-mfa* (Richardson, 2019) implementation, an advancement of the stochastic gradient descent (SGD) based MFA used in Richardson and Weiss (2018). Due to the SGD optimization, *torch-mfa* introduces additional parameters, such as learning rate and batch size. We set the batch size to 256, consistent with Richardson and Weiss (2018), and determined the optimal learning rate to be 10^{-2} via a coarse grid search (prior to the experiments). We modified the function `sgd_mfa_train` in the `mfa.py` script to implement a convergence criterion similar to Equation (14) in the main text. The

8. <https://github.com/variational-sublinear-clustering/vamm>
 9. <https://github.com/variational-sublinear-clustering/vammdx>
 10. <https://github.com/variational-sublinear-clustering/emmi>
 11. <https://github.com/variational-sublinear-clustering/emmidx>

original function does a fixed number of iterations. Unlike em-MFA and v-MFA, where the free energy is computed for the entire data set during each E-step, *torch-mfa* updates parameters after each batch, making full data set evaluation impractical. Instead, we computed the free energy on a randomly selected subset of the data set after each epoch and used this for the convergence criterion in Equation (14) in the main text. The subset size was set to $10\times$ the batch size, and the computational time for this evaluation was excluded from the time measurement.

AFK-MC²: We provide an efficient implementation of the AFK-MC² seeding method (Bachem et al., 2016a) integrated into v-MFA. For all experiments, the Markov chain length was set to 10.

In the following, we describe the zero-shot denoising algorithms that we used for comparison with the v-MFA algorithm in Section 3.4. Note that for certain images, some algorithms failed to produce a meaningful denoised result, e.g., yielding almost uniformly gray outputs. To address this issue, we use the PSNR and SSIM values of the noisy image whenever the corresponding denoised image performs worse in both metrics.

Noise2Self: We used the official Noise2Self implementation (Batson and Royer, 2019b) and executed the algorithm with the same configuration as Lequyer et al. (2022a), i.e., we increased the number of iterations from 500 to 20 000. To apply Noise2Self to RGB images, each color channel is processed independently.

Noise2Void: We used the official Noise2Void implementation (Krull et al., 2019b), following the ‘denoising2D_RGB’ example, available in the respective GitHub repository. To be consistent with Lequyer et al. (2022a), we used a patch size of 64×64 , 100 epochs and 100 steps per epoch, a train batch size of 16 and a neighborhood radius of 5.

Self2Self: We used the official Self2Self implementation (Quan et al., 2020b) and executed the algorithm with the configuration of the ‘demo.denoising.py’ script available in the respective GitHub repository. On the Confocal data set, the default learning rate of 10^{-4} failed to yield meaningful results. Hence, on this data set, we lowered the learning rate to 10^{-5} .

Deep Image Prior: We used the official Deep Image Prior implementation (Ulyanov et al., 2018a) and executed the algorithm with the same configuration as Lequyer et al. (2022a), including the same network (‘skip-net’) architecture and a fixed number of 3000 iterations. For the parameter ‘reg_noise_std’, we used a fixed value of $1/30$, independent of the noise level.

Noise2Fast: We used the official Noise2Fast implementation (Lequyer et al., 2022b) and executed the algorithm following the ‘N2F.py’ script for grayscale images and the ‘N2F_4D.py’ script for RGB images, both available in the respective GitHub repository.

Pixel2Pixel: We used the official Pixel2Pixel implementation (Ma et al., 2025a) and executed the algorithm with the configuration of the ‘Pixel2Pixel_real.py’ script available in the respective GitHub repository. On the Confocal data set, the default learning rate of 10^{-3} failed to yield meaningful results (the output image was uniformly gray). Hence, on

Algorithm	Hardware		
	Device	Model name	Cores used
v-MFA, em-MFA, k -means+ FA	CPU	AMD Genoa EPYC 9554	16*
v-MFA, em-MFA (YFCC100M & denoising)	CPU	AMD Genoa EPYC 9554	64
<i>torch-mfa</i>	GPU	NVIDIA H100 94GB SXM	all
N2S, N2V, DIP, S2S, N2F, P2P, DivN	GPU	NVIDIA L40S 48GB	all
BM3D	CPU	AMD Genoa EPYC 9554	64

Table B1: Details on the utilized hardware for the different algorithms and experiments. The asterisk (*) marks execution of four instances in parallel, with each instance using 16 of the 64 available cores, i.e., four independent runs were executed at the same time.

this data set, we lowered the learning rate to 10^{-4} , which resulted in meaningful results for all but one images.

DivNoising: We used the official DivNoising implementation (Prakash et al., 2021a) and executed the algorithm with the configuration of the ‘Convallaria’ and ‘Mouse_nuclei’ example available in the respective GitHub repository. For certain images and noise levels in BSD68 and Set12, DivNoising produced almost uniformly gray outputs. In such cases, we use the PSNR and SSIM values of the noisy image, as explained above.

BM3D: We used the official BM3D Python software package (Mäkinen, 2024) and executed the algorithm using the functions ‘bm3d’ for grayscale images and ‘bm3d_rgb’ for RGB images.

B.1.2 HARDWARE

Table B1 provides a detailed description of the hardware used for the various algorithms and experimental setups. To achieve high utilization, we executed four instances of v-MFA, em-MFA, or k -means+ FA in parallel, with each instance using 16 of the 64 available cores, i.e., four independent runs were executed at the same time. The tool GNU `parallel` (Tange, 2021) was employed for job scheduling. Exceptions to this are the training of v-MFA and em-MFA on YFCC100M and the denoising benchmarks in Sections 3.3 and 3.4 in the main text, where we executed only one instance at a time, using all 64 available cores. We executed *torch-mfa* on an NVIDIA H100 GPU equipped with 96 GB of VRAM. Running *torch-mfa* on the CPU configurations used by the other algorithms resulted in significantly longer execution times (GPU ≈ 1 h vs. CPU (16 cores) > 10 h for a single epoch). Similarly, for the denoising benchmarks in Section 3.4 in the main text, we executed Noise2Self, Noise2Void, Deep Image Prior, Self2SSelf, Noise2Fast, Pixel2Pixel and DivNoising on an NVIDIA L40s GPU equipped with 48 GB of VRAM. BM3D was executed on the same CPU as v-MFA.

B.1.3 DATA SETS AND PREPROCESSING

In the following, we provide additional information on the data sets used in the numerical experiments. The main properties of the data sets are listed in Table B2. Unless otherwise stated, we use the official *train* and *test* splits, and the images are used in their original form without preprocessing. Note that the label information was not used in any of the experiments. The data sets listed in Table B3 are used for denoising benchmarks, where each image is processed individually by the denoising algorithms.

CIFAR-10: The CIFAR-10 data set (Krizhevsky, 2009) contains of 60 000 natural color images with size 32×32 , divided into 50 000 training images and 10 000 test images.

CelebA: The original CelebA data set (Liu et al., 2015) contains 202 599 color images of faces sized at 178×218 . We preprocessed the data to align, crop and resize the images to 64×64 , following Richardson and Weiss (2018). Additionally, we merged the *train* and *valid* splits to obtain more training data.

EMNIST: The EMNIST data set (Cohen et al., 2017) contains of small cropped grayscale images of handwritten digits and letters, sized at 28×28 . The images are split into different groups. We used the *byclass* split, which contains 697 932 training images and 116 323 test images. To avoid zero variances at the image edges, standard Gaussian noise (with zero mean and standard deviation of 1) was added to the images.

SVHN: Similar to the digits in EMNIST, the SVHN data set (Netzer et al., 2011) contains images of small cropped digits in the range from 0 to 9. But in contrast to EMNIST, these images are real, colored photographs capturing house numbers at a size of 32×32 . The SVHN data set is split into 73 257 training images (*train*), 26 032 test images (*test*) and 531 131 additional training images (*extra*). To obtain more training data, we merged the *extra* and *train* images.

YFCC100M: The original YFCC100M data set (Thomee et al., 2016) contains real-world images and videos. We used all available images; some of the original 99 171 688 images were missing or corrupted and thus not included. The remaining 99 155 298 images of various sizes were center-cropped and resized with bilinear interpolation to 32×32 . We randomly selected a test split of 4 957 765 samples (5% of the data set) and used the remaining 94 197 533 samples for training.

The following data sets are used in the denoising benchmarks.

Set12: The Set12 data set contains of 12 grayscale natural images, with 7 of these 12 images of size 256×256 and the remaining 5 images of size 512×512 .

BSD68: BSD68 (Martin et al., 2001) comprises 68 grayscale natural images with a resolution of 481×321 or 321×481 pixels.

Confocal: This microscopy data set is the same subset of Confocal microscopy images from the Fluorescence Microscopy Data set (FMD; Zhang et al., 2019) as used by Lequyer et al. (2022a). It contains 5 inherently noisy grayscale images of size 512×512 . To obtain (pseudo) ground-truth references, multiple captures of the same microscopy scene were averaged (for more details, we refer to Zhang et al., 2019).

data set	train samples	test samples	data dimension D	
CIFAR-10	50 000	10 000	$32 \times 32 \times 3 =$	3072
CelebA	182 637	19 962	$64 \times 64 \times 3 =$	12 288
EMNIST	697 932	116 323	$28 \times 28 \times 1 =$	784
SVHN	604 388	26 032	$32 \times 32 \times 3 =$	3072
YFCC100M	94 197 533	4 957 765	$32 \times 32 \times 3 =$	3072

Table B2: High-dimensional data sets used in our experiments in Sections 3.1 to 3.3. The data dimension is given in pixel width $\times$ pixel height $\times$ color channels.

data set	samples	data dimension D	
Set12 images 01 – 07	7	$256 \times 256 \times 1 =$	65 536
Set12 images 08 – 12	5	$512 \times 512 \times 1 =$	262 144
BSD68	68	$321 \times 481 \times 1 =$	154 401
Confocal	5	$512 \times 512 \times 1 =$	262 144
Div2K ‘Tiger’	1	$2040 \times 1644 \times 3 =$	10 061 280

Table B3: Data sets used in the denoising experiments in Section 3.4. The data dimension is given in pixel width $\times$ pixel height $\times$ color channels.

Div2K: As an example of a high resolution image, we include one image of the Div2K data set (Agustsson and Timofte, 2017) in the denoising benchmark, namely the image with ID ‘0010’. This RGB image was chosen prior to any experiments and has a size of 2040×1644 . We denote this image as ‘Tiger’.

B.2 Numerical Control Experiments

The following subsections report the results of additional control experiments, offering supporting or complementary insights to those discussed in Section 3 in the main text.

B.2.1 COMPARISON OF COMPONENT-TO-COMPONENT DISTANCE ESTIMATES

In this section, we compare our v-MFA algorithm, which uses the proposed definition of g_c based on the estimate $D_{c\tilde{c}}$ in Equation (12) in the main text, with a variant of v-MFA (referred to as v-MFA^{Eucl.}) that uses the definition of g_c introduced in Hirschberger et al. (2022), based on $d_{c\tilde{c}}^2$ in Equation (A10). In addition to the MFA model, we also consider isotropic GMMs (compare Equation (5) in Hirschberger et al., 2022) and their optimization using variational EM with g_c based on $D_{c\tilde{c}}$ (referred to as v-ISO) and with g_c based on $d_{c\tilde{c}}^2$ (referred to as v-ISO^{Eucl.}). v-ISO^{Eucl.} is the algorithm used in previous work (Hirschberger et al., 2022).

We numerically compare v-MFA and v-MFA^{Eucl.} to investigate the differences of the here derived approach to the previous approach. Additionally, we compare v-ISO to v-ISO^{Eucl.} to investigate potential advantages and disadvantages under the constraint of isotropic components.

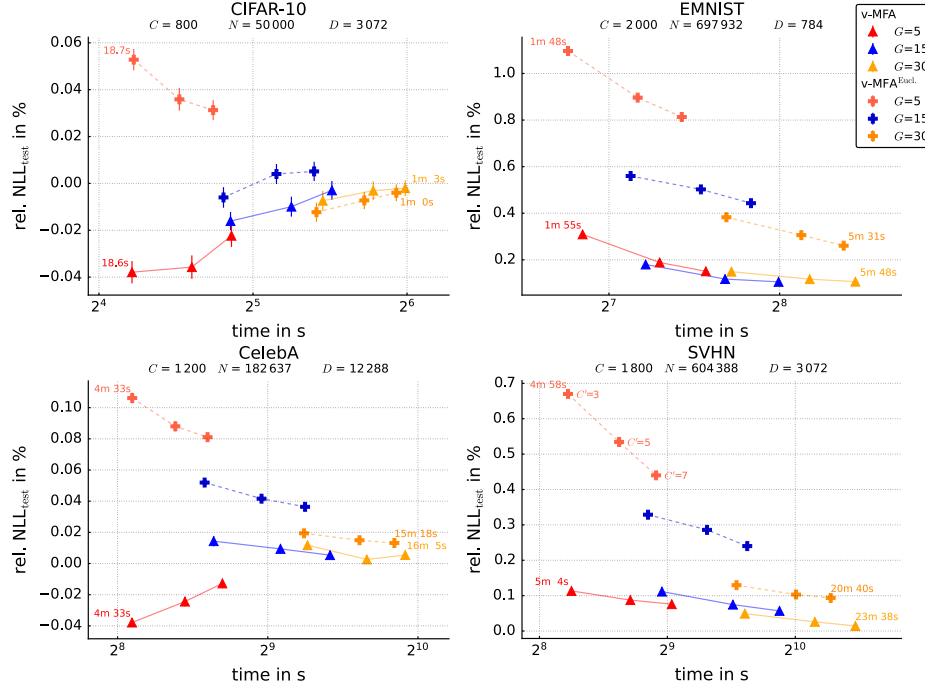


Figure B1: Effectiveness and efficiency of v-MFA and v-MFA^{Eucl.} in terms of NLL_{test} (relative to em-MFA) and the runtime. Triangles refer to v-MFA, while pluses denote v-MFA^{Eucl.}. Each of the four subfigures corresponds to experiments on one benchmark data set. The settings are similar to those in Figure 5 in the main text, but both algorithms are trained for the same number of iterations, as detailed in the text. For each configuration of $G \in \{5, 15, 30\}$, measurements refer to configurations with $C' \in \{3, 5, 7\}$ (three red, blue and orange triangles or pluses, respectively). Configurations with larger C' lie to the right, due to their longer runtimes. Each measurement point is averaged over 40 independent runs and error bars denote the standard error of the mean (SEM).

The models are initialized identically and using the same settings as described in Section 3.2 and Figure 5 in the main text. Since both methods use partial E-Steps with the same computational complexity, we run the algorithms for a fixed number of iterations. Concretely, we train v-MFA (or v-ISO) until convergence, and then train v-MFA^{Eucl.} (or v-ISO^{Eucl.}) for the same number of iterations.

The comparison results for the MFA models and for the isotropic GMMs are presented in Figure B1 and Figure B2, respectively. In the case of MFAs, the here derived v-MFA algorithm consistently performs better (or comparably within the error bars) in terms of NLL_{test} values when compared to v-MFA^{Eucl.} (especially for smaller search spaces $\mathcal{S}_n$). As the search space expands, the NLL_{test} values of both algorithms approach each other.

Also in the case of GMMs with isotropic components, v-ISO improves on NLL_{test} values compared to v-ISO^{Eucl.}, albeit at a slightly higher computational cost, due to smaller intersections in the search spaces. Notably, v-ISO^{Eucl.} only differs slightly from v-ISO in terms

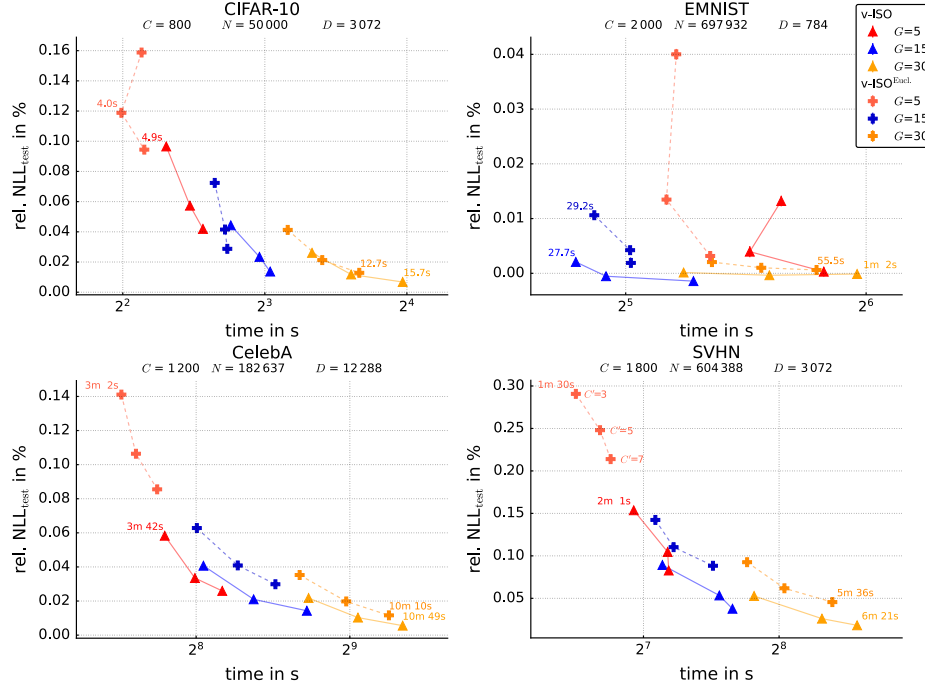


Figure B2: Effectiveness and efficiency of v-ISO and v-ISO^{Eucl.} in terms of NLL_{test} (relative to em-MFA) and the runtime. Triangles refer to v-ISO, while pluses denote v-ISO^{Eucl.}. Each of the four subfigures corresponds to experiments on one benchmark data set. The settings are similar to those in Figure 5 in the main text, but both algorithms are trained for the same number of iterations, as detailed in the text. For each configuration of $G \in \{5, 15, 30\}$, measurements refer to configurations with $C' \in \{3, 5, 7\}$ (three red, blue and orange triangles or pluses, respectively). Each measurement point is averaged over 40 independent runs and error bars denote the standard error of the mean (SEM).

of NLL_{test} values for most of the data sets and settings of C' and G (see Figure B2), which confirms the suitability d_{cc}^2 of Equation (A10) for the isotropic case, for which it was originally derived (Hirschberger et al., 2022). In contrast, in the case of GMMs with flexible covariances (i.e., for the MFA model), the significant improvements of v-MFA compared to v-MFA^{Eucl.} highlight the importance of the more general definition of g_c using D_{cc} given by Equation (12) in the main text.

B.2.2 COMPARISON TO OTHER GMM/MFA APPROACHES

In experiments in Section 3.1 and Section 3.2 in the main text, we showed the speed-up of our variationally accelerated algorithm (v-MFA) compared to conventional EM optimization for MFA (em-MFA). Other variants of GMMs and MFAs and corresponding optimization methods do exist but often have focus on areas of application other than scalability. For instance, the AECM (McLachlan et al., 2003) approach and the ECM approach (Zhao and Yu, 2008) have been applied to MFA optimization. Both approaches do not focus

on scalability and, as discussed in Appendix A.1, they exhibit a higher computational complexity compared to conventional EM (and we compare to the latter). Methods such as Hirschberger et al. (2022) and Exarchakis et al. (2022) address scalability but are limited to isotropic or diagonal covariances. A method by Asheri et al. (2021) allows for more general covariance constraints, but their focus is on enabling larger numbers of components for the same amount of data. Their work does not focus on improving scalability compared to conventional GMM optimization. The method in the literature most related to our approaches and our goal of scalability is thus *torch-mfa* which makes use of stochastic gradient decent integrated as part of the PyTorch framework.

The *torch-mfa* algorithm represents a further advancement of the TensorFlow implementation employed to produce results previously reported by the same group (Richardson and Weiss, 2018) (for further details, see Appendix B.1.1). The work by Richardson and Weiss (2018) documents (to the knowledge of the authors) the largest MFA models prior to this study. However, at the scales here addressed, one can observe clear efficiency limits of *torch-mfa* (as well as its predecessor implementation), which we elaborate on further below. When considering time measurements for the various settings, note that although actual runtimes have significant practical relevance, they are inherently dependent on implementation details and hardware. We do not claim, nor do we expect, that *torch-mfa* is optimal in any specific sense, but rather representative of what a typical implementation might achieve.

In addition to *torch-mfa*, we also compare against GMMs constrained to diagonal covariance matrices. Like *torch-mfa*, v-MFA models intra-component correlations. A comparison with GMMs with diagonal covariance matrices can thus be instructive about the benefits of modeling correlations. For our comparison, we concretely consider diagonal GMMs trained via conventional EM (denoted em-DIAG) and via our variational approach (denoted v-DIAG).

To assess the performance of *torch-mfa*, em-DIAG and v-DIAG against v-MFA and em-MFA, we adopted the configurations of MFA hyperparameters used in an experiment on the CelebA data set (see Richardson and Weiss, 2018). That is, the number of components and the hyperplane dimension were set to $C = 1000$ and $H = 10$ for all algorithms using MFAs. We ran *torch-mfa* on a GPU with the configurations specified in Appendix B.1. This is in contrast to the CPU configurations for the other algorithms, as discussed in Appendix B.1.2. The experiments in Richardson and Weiss (2018) were performed utilizing hierarchical training (for more details we refer to Richardson and Weiss, 2018). To ensure a fair comparison, we trained *torch-mfa* in our experiments directly on the complete data set, without hierarchical training. In addition to these algorithms, we used the k -means+FA algorithm in our experimental comparison.

The results of our comparison are shown in Table B4. As diagonal GMMs contain substantially fewer parameters than MFAs with the same number of components C , Table B4 also contains v-DIAG versions with larger C . For $C = 6000$, v-DIAG contains approximately the same number of parameters as v-MFA with $C = 1000$. For $C = 10\,000$ and $C = 20\,000$, v-DIAG far exceeds the number of parameters of v-MFA. The number of components for v-DIAG with $C > 1000$ is indicated by superscripts on v-DIAG in Table B4.

	NLL_{test}	rel. NLL_{test}	runtime	joint / distance eval.	iterations
em-MFA	57571 ± 3	0.00 %	1h 16m $\pm$ 53.5s	$(4.66 \pm 0.05) \cdot 10^9$	25.5
v-MFA	57565 ± 2	$(-0.01 \pm 0.01) \%$	9m 2s $\pm$ 11.6s	$(2.47 \pm 0.05) \cdot 10^8$	35.0
k -means+ FA	58373 ± 2	$(1.39 \pm 0.00) \%$	12m 12s $\pm$ 2.8s	$(1.83 \pm 0.01) \cdot 10^{10}$	106.7
<i>torch-mfa</i>	57868 ± 13	$(0.52 \pm 0.02) \%$	1d 14h $\pm$ 1h 35m	$(6.8 \pm 0.3) \cdot 10^9$	38.5
em-DIAG	61232 ± 5	$(6.36 \pm 0.01) \%$	5m 13s $\pm$ 0.5s	$(2.56 \pm 0.01) \cdot 10^9$	14.0
v-DIAG	61257 ± 5	$(6.40 \pm 0.01) \%$	1m 5s $\pm$ 0.7s	$(1.69 \pm 0.02) \cdot 10^8$	24.0
v-DIAG ^{6k}	60876 ± 3	$(5.74 \pm 0.01) \%$	2m 38s $\pm$ 0.6s	$(3.87 \pm 0.02) \cdot 10^8$	27.5
v-DIAG ^{10k}	60864 ± 4	$(5.72 \pm 0.00) \%$	4m 24s $\pm$ 1.5s*	$(7.12 \pm 0.02) \cdot 10^8$	28.0
v-DIAG ^{20k}	60930 ± 1	$(5.83 \pm 0.01) \%$	12m 26s $\pm$ 1.0s*	$(2.21 \pm 0.01) \cdot 10^9$	27.0

Table B4: Comparison of em-MFA, v-MFA, k -means+ FA , *torch-mfa*, em-DIAG with $C = 1000$ and multiple v-DIAG variants with $C \in \{1000, 6000, 10\,000, 20\,000\}$ on CelebA. To ensure consistency with the other methods, we define an iteration for *torch-mfa* as completed when the model has processed the entire data set, rather than just one data batch. Results are averaged over 4 independent runs. All errors indicate the standard error of the mean (SEM), rounded up to the displayed precision. The best values for NLL_{test} , the relative NLL_{test} , total runtime including initialization (d=days, h=hours, m=minutes, s=seconds) and the number of joint or distance evaluations are marked bold *(initialization starts dominating the runtime for $C = 10\,000$ and $20\,000$, e.g., the initialization alone already requires over 10m for $C = 20\,000$).

The findings show that v-MFA achieved the lowest NLL_{test} value in comparison to the other algorithms, with NLL values that are essentially equal to those of em-MFA. Among the MFA-based approaches, v-MFA was clearly the fastest algorithm, completing in approximately 9 minutes. Following closely behind is k -means+ FA , with a runtime of approximately 12 minutes. The em-MFA algorithm exhibited considerably worse runtime performance, necessitating over 1 hour to complete. Remarkably, the *torch-mfa* algorithm was the most time-consuming, requiring approximately one and a half days to complete, making it the slowest among the algorithms.

A possible reason for the long runtimes of *torch-mfa* could be related to the optimization procedure. Unlike the other algorithms, which update model parameters once per iteration (M-step evaluated over the entire data set), *torch-mfa* updates parameters after each data batch. Consequently, this results in approximately $\frac{N}{\text{batch size}} = \frac{182637}{256} \approx 713\times$ as many (yet less computationally expensive) parameter updates. In this context, we attempted to increase the batch size (to 512) in order to potentially reduce the runtime. However, this was not possible due to memory constraints of the used GPU. Another factor to consider is that the design objectives of the em-MFA and v-MFA algorithms were mainly focused on optimizing speed, whereas speed might not have been the main objective for the *torch-mfa* algorithm.

In terms of the number of joint or distance evaluations, v-MFA also exhibited an order of magnitude fewer evaluations than em-MFA and *torch-mfa*, which were within the same order of magnitude. The greatest number of distance evaluations was performed by k -means+ FA , once more an order of magnitude higher than those of em-MFA and *torch-mfa*.

This observation seems to be inconsistent with the runtimes of the algorithms. However, a distance evaluation of k -means is significantly less computationally expensive than a joint evaluation of the other algorithms.

When comparing the MFA-based approaches to GMMs with diagonal covariance matrices, it can be observed that none of the diagonal GMM variants could reach NLL_{test} values as good as the MFA-based models. Increasing the number of components improves performance to some extent, but even for $C = 10\,000$, the NLL_{test} value remains significantly higher than those of the MFA-based approaches (although v-DIAG has more model parameters available in this case). Note that further increasing the number of components does not improve performance, because v-DIAG starts to overfit the training data. NLL_{test} values for v-DIAG with C increased to 20 000 therefore get worse. For this reason, using $C = 10\,000$ achieves the best NLL_{test} value that we have observed for v-DIAG.

B.2.3 CONTROL EXPERIMENTS FOR THE v-MFA SCALABILITY ANALYSIS

The results in Figure 4 in the main text do not account for the quality of em-MFA and v-MFA. However, it is important to consider the optimization quality also for the investigation of scaling behavior because the improved scaling of v-MFA may be attributed to two key factors: First, the variational approach reduces the computational complexity of the algorithm, which is the desired effect we aim to measure. Second, some loss in quality can typically be expected for v-MFA as it represents an approximation of the conventional EM optimization of MFA.

Therefore, we conducted controls by repeating the experiments in Section 3.1 in the main text, now incorporating the different optimization qualities as measured by the NLL on the test data sets. Concretely, we stopped the em-MFA algorithm once it achieved the same NLL_{test} as was achieved after a full run of the v-MFA algorithm.

As we only evaluate NLL_{test} values after a full iteration, values for em-MFA and v-MFA never match precisely, however. To address this, we proceeded as follows: We trained the em-MFA algorithm while monitoring the NLL_{test} after each iteration during training. When the NLL_{test} of em-MFA surpassed that of v-MFA, we stopped the algorithm. We then used the model parameters from the *previous* iteration of em-MFA to ensure that the quality of em-MFA was approximately the same as (but slightly worse than) the quality of v-MFA. The results of this procedure are denoted as $\text{em-MFA}^\dagger$ in Figure B3, i.e., $\text{em-MFA}^\dagger$ shows the speed-up achievable if em-MFA is only required to reach the final optimization quality of v-MFA. All other results shown in Figure B3 are taken from Figure 4 in the main text. For two of the four data sets, $\text{em-MFA}^\dagger$ achieved a smaller scaling exponent than em-MFA. The number of iterations required to obtain the same NLL_{test} as v-MFA decreases slightly with increasingly large values of C . For both CIFAR-10 and CelebA, the scaling exponents of $\text{em-MFA}^\dagger$ and em-MFA are approximately the same. On CIFAR-10, the NLL_{test} values of v-MFA are lower than those of em-MFA for sufficiently large values of C , as also observed in Figure 5 in the main text. Consequently, the algorithm requires more iterations than previously to reach the desired quality.

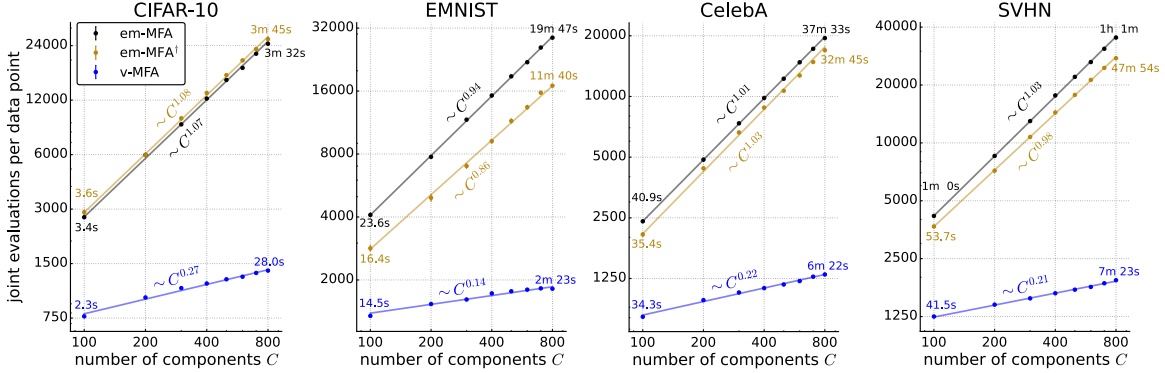


Figure B3: Scaling behavior accounting for quality. Depicted is the number of joint evaluations per data point for increasingly large numbers of components C . The values for v-MFA and em-MFA are the same as in Figure 4 in the main text. em-MFA[†] denotes the results for em-MFA trained until it reaches approximately the same NLL_{test} as v-MFA (see text for details). Results are averaged over 40 independent runs, with error bars indicating standard errors of the mean (SEM). Note that the SEM values are very small and barely visible in the plot. The legend in the first subfigure applies to all four subfigures. Lines represent regressions used to determine the scaling exponent of C as in Figure 4 in the main text, while annotations provide information on total training time.

Overall, the scaling exponent of v-MFA remains significantly lower than that of em-MFA[†] across all four data sets. The control experiments thus show that the improved scaling behavior of v-MFA can be attributed to the sublinear complexity of the algorithm, while the impact of reduced quality contributes only a small fraction to the effect or is negligible.

B.2.4 ABLATION STUDY FOR APPROXIMATION QUALITY

The v-MFA algorithm is a variational approximation method that emphasizes computational efficiency, potentially at the expense of approximation quality (compare, e.g., EMNIST or SVHN in Figure 5). Its performance is influenced by: (1) the degree to which the assumption of truncated posteriors fits the underlying data, and (2) the effectiveness of the variational EM procedure in optimizing the variational parameters $\mathcal{K}_n$.

To disentangle the contributions of these two factors to the performance of v-MFA, we conducted an ablation study. In addition to the experiments reported in Section 3.2 in the main text, we evaluated v-MFA on the same data sets and the same values for C' . But instead of using the efficiently computed sets $\mathcal{K}_n$ of v-MFA, we used the optimal values $\mathcal{K}_n^{\text{opt}}$, as defined in Equation (7). Finding $\mathcal{K}_n^{\text{opt}}$ is computationally as demanding as an E-Step of em-MFA. However, using $\mathcal{K}_n^{\text{opt}}$ allows a quantification of the effect of truncation, without the influence of other approximations. The residual difference in quality therefore reflects the accuracy of our approximation in estimating suitable variational parameters. The results are presented in Figure B4.

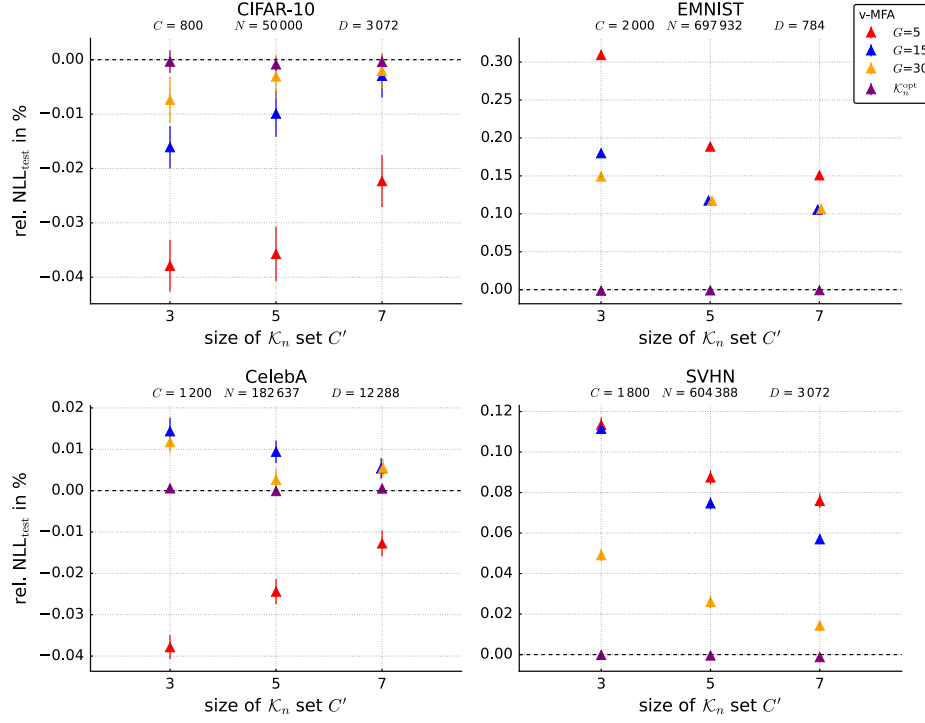


Figure B4: Effectiveness in terms of negative log-likelihood (NLL) of v-MFA for different configurations of C' and G , as well as for a variant of v-MFA optimized using the optimal variational parameters $\mathcal{K}_n^{\text{opt}}$ (as defined in Equation 7). Each of the four subfigures refers to experiments on one benchmark data set. The y-axes denote the relative NLL on the test set w.r.t. em-MFA as baseline, given by Equation (15). Each measurement point is averaged over 40 independent runs, with error bars indicating standard error of the mean (SEM) (note that the SEM values may be smaller than the symbol sizes).

Considering the figure, it can be observed that when the v-MFA algorithm is evaluated using the optimal variational parameters $\mathcal{K}_n^{\text{opt}}$, its performance is almost indistinguishable from that of em-MFA, already for $C' = 3$. Consequently, the effect of truncation is (almost) negligible, i.e., truncation itself does not account for the observed differences between v-MFA and em-MFA. Figure B4 shows that the differences (at least for the studied data sets) are mainly due to the variational optimization procedure finding sets $\mathcal{K}_n$ different from the optimal sets $\mathcal{K}_n^{\text{opt}}$. This, in turn, leads to different model parameter optimizations due to the differences in the $\mathcal{K}_n$ sets. Notably, the non-optimal $\mathcal{K}_n$ sets of v-MFA do not necessarily result in worse optimization results compared to em-MFA. In some cases, e.g., for CIFAR-10 and CelebA, v-MFA with $G = 5$ even results in slightly better NLL_{test} values compared to the (much more computationally expensive) em-MFA optimization. This effect is likely due to alternative learning trajectories, e.g., avoiding overfitting or some local optima.

B.2.5 ABLATION STUDY FOR DIFFERENT INITIALIZATIONS

In the experiments reported in the main text, we consistently used the same default initialization strategy for the variational and model parameters (except in Section 3.3). This initialization strategy that leverages AFK-MC², is described in Section 2.4. A crucial aspect of AFK-MC² is that it provides well-chosen seedings while maintaining low computational complexity. To systematically analyze the impact of initialization on overall performance, we further investigate and compare AFK-MC² against alternative initialization strategies of varying sophistication. Specifically, we consider initialization using random noise, uniform sampling, k -means++, and the k -means+FA algorithm, as detailed below. For all here investigated initialization strategies, the sets g_c are initialized using the default procedure and *warm-up* iterations are performed, as described in Section 2.4.

Noise: The component means are initialized by sampling uniformly at random within the range of the observed data for each dimension, i.e., $\mu_{cd} \sim \mathcal{U}\{\min(x_{1:N,d}), \max(x_{1:N,d})\}$. The factor loading matrices are initialized using the default strategy (cf. Section 2.4). The diagonal covariance matrices are set to the identity, i.e., $D_c = I$. The sets $\mathcal{K}_{1:N}$ and $g_{1:C}$ are initialized by sampling indices uniformly from $\{1, \dots, C\}$ (ensuring uniqueness of the indices within each set).

Uniform and k -means++: These initialization strategies closely follow that used in the main experiments (see Section 3). The factor loadings, diagonal matrix and g_c set are initialized as described in Section 2.4. The component means are initialized by sampling uniformly from the data points, i.e., $\mu_c = \mathbf{x}_n$ with $n \sim \mathcal{U}\{1, N\}$ (ensuring uniqueness of the indices), or by selecting data points via k -means++ seeding (Arthur and Vassilvitskii, 2007). In both cases, the corresponding component index is inserted into the set $\mathcal{K}_n$. Subsequently, $\mathcal{K}_{1:N}$ are populated with further indices by sampling uniformly from $\{1, \dots, C\}$ (while ensuring uniqueness of the indices). The Uniform strategy was also applied in Section 3.3.

k -means+FA: This initialization strategy is the most sophisticated one. First, we run the k -means+FA algorithm as described in Appendix B.1.1. The model parameters $\Theta = \{\pi_{1:C}, \Lambda_{1:C}, \mu_{1:C}, D_{1:C}\}$ are then initialized using the parameters obtained from the k -means+FA algorithm. For each set $\mathcal{K}_n$, we insert the index of the closest cluster (component) c for the data point $\mathbf{x}_n$ as identified by the k -means algorithm (compare Equation B1). Subsequently, $\mathcal{K}_{1:N}$ are populated with additional indices sampled uniformly from $\{1, \dots, C\}$ (ensuring uniqueness of the indices).

Due to less informed initialization methods, we observed that certain components occasionally become empty during training, i.e., they contain no data points, causing their prior to become zero. To maintain a fixed number of components (allowing for a fair comparison), we address this issue by splitting an existing component (cf. Caron et al., 2018; Monnier et al., 2020). Specifically, when a component c becomes empty, we sample another component c' from the prior distribution, such that larger components are more likely to be selected. The parameters of the empty component c are then re-initialized based on those of the sampled component c' , with additional small perturbations. Finally, we insert c into $g_{c'}$, which allows the $\mathcal{K}_n$ sets to be updated accordingly in the next E-Step.

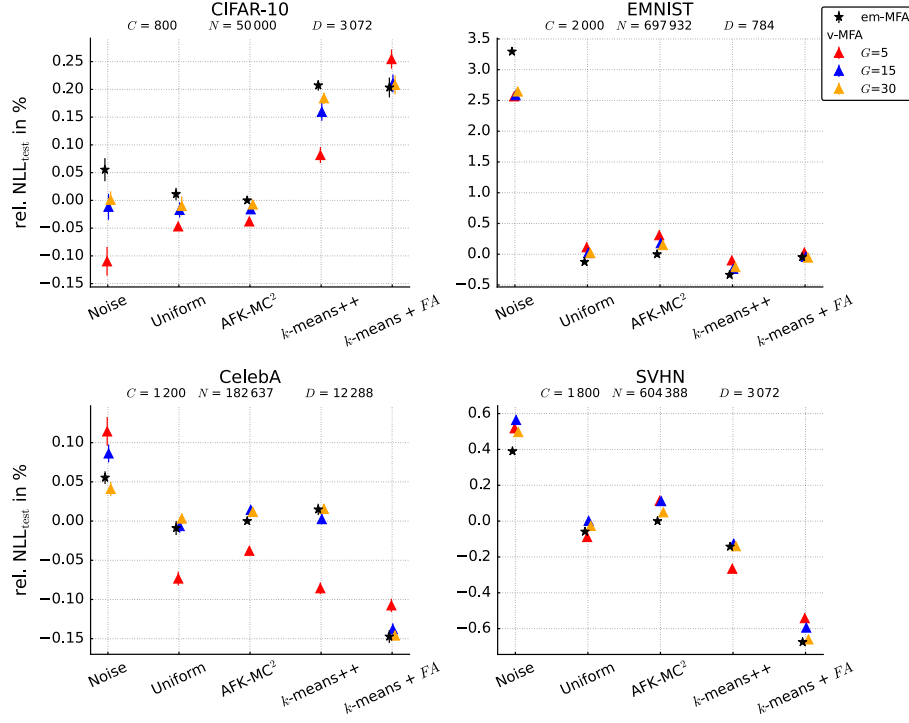


Figure B5: Comparison of different initialization strategies. Each of the four subfigures refers to experiments on one benchmark data set. The y-axes denote the relative NLL on the test set w.r.t. em-MFA initialized which AFK-MC², given by Equation (15). Each measurement point is averaged over 40 independent runs, with error bars indicating standard error of the mean (SEM) (note that the SEM values may be smaller than the symbol sizes).

The comparison of the different initialization strategies is presented in Figure B5. The results indicate that more sophisticated methods, such as *k-means++* or *k-means+FA*, can yield improved NLL_{test} values for both v-MFA and em-MFA, but not in all cases. However, this improvement comes at the expense of increased computational complexity. In particular, the *k-means+FA* algorithm alone requires, for most settings, much more time than an entire v-MFA optimization with AFK-MC² seeding (see Figure 5 in the main text).

Using uniformly sampled data points as seeds for components (the ‘Uniform’ method), as also employed in Section 3.3, yields high final optimization quality that is similar to that obtained with AFK-MC² seeding, which may be taken as evidence for a substantial degree of robustness of the v-MFA algorithm w.r.t. initialization methods. Only when initializing components with noise (the least informed method considered), performance decreases in almost all settings. Finally, note that variations in NLL_{test} values across different initializations are comparable for both v-MFA and em-MFA, i.e., the used variational optimization does not result in a noticeable increase of the sensitivity to initialization.

B.3 Additional Information on the Quality Analysis

Additional information and results about the experiments in Section 3.2 in the main text are provided in Table B5. The table reports the relative NLL_{test} , the average speed-up in terms of runtime and joint evaluations (or distance evaluations for k -means+ FA) compared to em-MFA as well as the median number of iterations until convergence. Additionally, for the v-MFA algorithm, the *warm-up* iterations are reported (note that the total number of iterations includes the *warm-up* iterations).

B.4 Additional Denoising Results

In the following, we show additional denoising results. Visual comparisons of the denoised images produced by various blind zero-shot denoising algorithms are presented in Figure B6. Further examples are available online.¹²

We also include other noise levels for Set12 and the ‘Tiger’ image from Div2K and compare v-MFA against other variants of GMM-based denoising. These include conventional EM training with em-MFA instead of the variational optimization, as well as variants where the MFA model is replaced by a GMM with diagonal covariances (denoted em-DIAG and v-DIAG). These results are presented in Table B6. Furthermore, we performed numerical experiments with different hyperplane dimensions H of the MFA model (see Table B7).

The results from Table B6 can be summarized as follows. While em-MFA shows in some settings marginal improvements in denoising performance over v-MFA, it comes at the cost of substantially increased runtimes (usually between one and two orders of magnitude longer runtimes). A similar behavior is observed for em-DIAG compared to v-DIAG but runtime differences are not as pronounced. Regarding the comparison between the MFA model and diagonal GMMs, the MFA-based algorithms consistently show higher PSNR values (also compare Table B4).

For the results in Section 3.4 and Table B6, we employed a fixed set of hyperparameters across all data sets and noise levels. However, these hyperparameters are expected to influence denoising performance. For the number of components, this is already evident from Figure 8, where we increased the number of components up to $C = 10\,000$. A low hyperplane dimensionality (we chose $H = 5$ for the experiments in Section 3.4) was observed to result in a favorable trade-off between optimization qualities and runtimes. Here we further investigate the effect of the hyperplane dimensionality H . Concretely, we repeated the experiment from Section 3.4 for v-MFA, varying the hyperplane dimension over $H \in \{3, 5, 10, 20\}$. Increasing H allows each component to model more intra-component correlations but model sizes and computational costs increase with higher H .

As can be observed by inspecting the results of Table B7, increasing H continuously increases performance for the high-resolution image of the Div2K data set. However, performance does not necessarily increase, and it decreases for the Set12 data set, for instance. This observation can be explained by limited data available for estimating the model parameters. For Set12, there is much less information available than for the Div2K image. If H is increased, more model parameters have to be estimated, and data variations due to noise

12. <https://github.com/variational-sublinear-clustering/vammdx>

Data set #Components	Algorithm Name	C'	G	rel. NLL_{test}	Rel. speed-up		#Iterations	
					Time	Joints/Dist.	warm-up	total
CIFAR-10 $C = 800$	em-MFA	-	-	0.0%	1.0×	1.0×	-	31.0
	k -means+ FA	-	-	1.64%	4.4×	0.5×	-	77.6
	v-MFA	3	5	-0.04%	11.5×	29.8×	15.0	52.0
		3	15	-0.02%	7.4×	17.2×	7.0	39.0
		3	30	-0.01%	4.9×	10.7×	5.0	37.0
		5	5	-0.04%	8.8×	22.3×	12.0	45.5
		5	15	-0.01%	5.6×	12.9×	6.0	37.0
		5	30	-0.00%	3.9×	8.4×	4.0	35.5
		7	5	-0.02%	7.3×	18.2×	10.0	42.0
		7	15	-0.00%	4.7×	10.6×	6.0	37.0
		7	30	-0.00%	3.4×	7.2×	4.0	35.0
EMNIST $C = 2000$	em-MFA	-	-	0.0%	1.0×	1.0×	-	29.0
	k -means+ FA	-	-	7.48%	5.5×	0.3×	-	112.6
	v-MFA	3	5	0.31%	19.3×	41.9×	37.0	87.0
		3	15	0.18%	14.9×	30.1×	16.0	48.0
		3	30	0.15%	10.5×	20.7×	8.0	38.5
		5	5	0.19%	14.1×	30.6×	31.0	75.0
		5	15	0.12%	10.8×	21.8×	13.0	43.0
		5	30	0.12%	7.7×	15.0×	8.0	37.0
		7	5	0.15%	11.7×	25.2×	27.0	67.0
		7	15	0.11%	8.7×	17.4×	11.0	40.0
		7	30	0.11%	6.4×	12.4×	6.0	35.0
CelebA $C = 1200$	em-MFA	-	-	0.0%	1.0×	1.0×	-	24.0
	k -means+ FA	-	-	1.03%	3.3×	0.2×	-	108.6
	v-MFA	3	5	-0.04%	11.8×	30.7×	20.0	57.0
		3	15	0.01%	8.1×	20.3×	8.0	35.0
		3	30	0.01%	5.3×	12.9×	6.0	31.0
		5	5	-0.02%	9.3×	23.5×	15.0	48.0
		5	15	0.01%	6.0×	14.7×	6.0	32.0
		5	30	0.00%	4.0×	9.7×	5.0	29.0
		7	5	-0.01%	7.8×	19.5×	13.0	42.0
		7	15	0.01%	4.7×	11.6×	6.0	30.5
		7	30	0.01%	3.3×	8.1×	4.0	28.0
SVHN $C = 1800$	em-MFA	-	-	0.0%	1.0×	1.0×	-	42.0
	k -means+ FA	-	-	3.19%	7.0×	0.4×	-	110.3
	v-MFA	3	5	0.11%	25.6×	61.9×	17.0	78.0
		3	15	0.11%	15.7×	34.3×	8.0	54.0
		3	30	0.05%	10.0×	21.2×	6.0	50.0
		5	5	0.09%	18.6×	44.6×	13.0	68.0
		5	15	0.07%	10.7×	23.3×	7.0	51.0
		5	30	0.03%	6.8×	14.5×	6.0	49.0
		7	5	0.08%	14.9×	35.4×	11.0	63.0
		7	15	0.06%	8.3×	18.0×	6.0	50.0
		7	30	0.01%	5.5×	11.6×	5.0	47.0

Table B5: Additional information and results for the experiments in Section 3.2 in the main text. Standard errors of the mean (SEM) for relative NLL_{test} were below 0.01%.



Figure B6: Visual comparisons of several denoising algorithms applied to images from Set12, BSD68, Confocal and the ‘Tiger’ image from Div2K (from top to bottom). All images were corrupted with additive Gaussian noise with a noise level of $\sigma = 50$, except for Confocal, which has inherent (natural) noise. PSNR and SSIM (10 $\times$) values are displayed in the bottom right corner of the noisy and denoised images.

are being modeled with available parameters (a form of overfitting). Finally, we observe that lower noise levels allow for higher H , which can also be explained by more limited amounts of information being available for higher noise levels.

Data set	σ	non-blind zero-shot			blind zero-shot									
		BM3D	DivN	N2S	N2V	DIP	S2S	N2F	P2P	v-MFA	em-MFA	v-DIAG	em-DIAG	
15	PSNR	32.39 ± 1.13	28.35 ± 2.05	28.45 ± 2.86	29.09 ± 1.43	28.42 ± 1.97	32.08 ± 1.21	31.15 ± 1.34	30.26 ± 1.79	30.59 ± 1.73	30.74 ± 1.59	26.91 ± 2.07	27.09 ± 1.98	
	SSIM	8.96 ± 0.22	8.14 ± 0.76	7.51 ± 1.19	8.27 ± 0.27	8.00 ± 0.78	8.89 ± 0.26	8.72 ± 0.22	8.66 ± 0.25	8.67 ± 0.35	8.67 ± 0.33	7.90 ± 0.52	7.94 ± 0.51	
	Time	0.80 ± 0.33	298.69 ± 34.37	1136.35 ± 706.53	296.43 ± 24.83	38.06 ± 1.26	3251.98 ± 2221.42	3.85 ± 2.22	17.07 ± 10.56	1.50 ± 1.09	35.47 ± 25.95	0.46 ± 0.23	2.61 ± 1.55	
Set12	PSNR	29.99 ± 1.29	26.95 ± 1.63	27.15 ± 2.47	27.35 ± 1.55	27.16 ± 1.65	30.03 ± 1.13	29.06 ± 1.16	28.80 ± 1.40	29.15 ± 1.40	29.19 ± 1.30	26.63 ± 1.88	26.73 ± 1.87	
	SSIM	8.52 ± 0.30	7.80 ± 0.52	7.37 ± 1.28	7.53 ± 0.29	7.74 ± 0.53	8.49 ± 0.32	8.22 ± 0.30	8.11 ± 0.31	8.25 ± 0.33	8.18 ± 0.35	7.78 ± 0.50	7.81 ± 0.48	
	Time	0.81 ± 0.33	283.20 ± 49.93	1139.62 ± 708.82	296.50 ± 27.94	37.60 ± 1.19	3262.46 ± 2218.07	3.32 ± 2.05	16.94 ± 10.93	1.37 ± 0.80	36.06 ± 23.60	0.46 ± 0.22	2.07 ± 1.21	
35	PSNR	28.42 ± 1.31	25.37 ± 2.76	25.08 ± 2.87	25.71 ± 1.17	25.54 ± 1.16	28.44 ± 1.10	27.48 ± 1.08	27.47 ± 1.13	27.81 ± 1.27	27.82 ± 1.19	26.16 ± 1.76	26.30 ± 1.70	
	SSIM	8.15 ± 0.35	7.16 ± 1.33	6.49 ± 1.48	6.71 ± 0.56	7.05 ± 0.52	8.09 ± 0.33	7.76 ± 0.36	7.49 ± 0.34	7.76 ± 0.34	7.62 ± 0.39	7.59 ± 0.50	7.62 ± 0.48	
	Time	0.81 ± 0.33	245.19 ± 66.98	1135.72 ± 707.89	299.21 ± 26.24	37.75 ± 1.47	3254.99 ± 2220.10	3.12 ± 2.29	16.99 ± 10.98	1.38 ± 0.90	30.17 ± 18.40	0.47 ± 0.18	1.99 ± 1.09	
50	PSNR	26.75 ± 1.27	23.81 ± 3.23	23.66 ± 3.15	24.02 ± 1.57	22.75 ± 0.91	26.54 ± 1.07	25.89 ± 1.07	25.72 ± 0.90	26.12 ± 1.19	26.05 ± 1.14	25.32 ± 1.56	25.44 ± 1.53	
	SSIM	7.68 ± 0.41	6.66 ± 1.49	6.17 ± 1.49	5.80 ± 0.67	5.48 ± 0.55	7.42 ± 0.37	7.24 ± 0.42	6.50 ± 0.40	7.03 ± 0.39	6.77 ± 0.50	7.25 ± 0.51	7.27 ± 0.47	
	Time	0.83 ± 0.35	197.02 ± 69.79	1140.80 ± 712.30	294.17 ± 27.80	37.91 ± 1.39	3238.05 ± 2197.54	3.36 ± 2.86	17.08 ± 10.95	1.23 ± 0.73	34.93 ± 23.30	0.44 ± 0.19	1.73 ± 0.92	
25	PSNR	28.61 ± 2.52	25.11 ± 3.25	26.72 ± 3.23	26.30 ± 2.67	25.96 ± 2.84	28.46 ± 2.85	28.11 ± 2.44	27.26 ± 2.89	27.22 ± 3.33	27.29 ± 3.23	25.36 ± 3.46	25.48 ± 3.41	
	SSIM	8.04 ± 0.66	6.60 ± 1.67	7.38 ± 0.92	7.06 ± 0.70	7.13 ± 0.79	7.97 ± 0.72	7.87 ± 0.61	7.50 ± 0.59	7.54 ± 0.81	7.53 ± 0.72	6.89 ± 1.05	6.94 ± 1.03	
	Time	0.79 ± 0.02	224.38 ± 94.58	1198.04 ± 2.60	920.73 ± 535.47	39.68 ± 0.48	3230.29 ± 56.55	3.67 ± 0.90	20.97 ± 0.19	1.42 ± 0.32	27.59 ± 2.09	0.44 ± 0.05	2.16 ± 0.25	
BSD68	PSNR	25.67 ± 2.60	20.72 ± 4.89	24.49 ± 2.26	23.22 ± 2.78	22.28 ± 1.62	25.70 ± 2.48	25.31 ± 2.35	24.84 ± 2.10	25.30 ± 2.66	25.22 ± 2.42	24.35 ± 3.37	24.43 ± 3.35	
	SSIM	6.89 ± 0.91	5.14 ± 2.02	6.39 ± 0.80	5.16 ± 0.77	5.16 ± 0.98	6.87 ± 0.86	6.72 ± 0.80	5.91 ± 0.51	6.51 ± 0.86	6.31 ± 0.67	6.32 ± 1.18	6.37 ± 1.15	
	Time	0.81 ± 0.03	157.14 ± 96.25	1185.39 ± 2.74	935.96 ± 552.42	39.99 ± 0.83	3222.79 ± 61.80	3.36 ± 0.97	21.14 ± 0.33	1.40 ± 0.24	25.95 ± 1.14	0.46 ± 0.07	1.97 ± 0.28	
Confocal	PSNR	–	–	36.20 ± 3.88	35.70 ± 3.53	34.53 ± 3.89	36.51 ± 3.39	36.60 ± 3.84	34.02 ± 2.33	36.83 ± 3.54	36.84 ± 3.52	35.64 ± 3.52	35.65 ± 3.52	
	SSIM	–	–	9.26 ± 0.38	9.18 ± 0.45	8.94 ± 0.64	9.28 ± 0.38	9.33 ± 0.39	9.04 ± 0.48	9.34 ± 0.37	9.34 ± 0.37	9.14 ± 0.51	9.14 ± 0.51	
	Time	–	–	1964.20 ± 1.22	203.80 ± 20.56	41.93 ± 3.54	5884.49 ± 140.50	8.42 ± 0.99	30.71 ± 2.24	4.43 ± 1.63	89.62 ± 37.24	1.35 ± 0.66	8.94 ± 3.86	
25	PSNR	32.16	29.00	30.02	29.59	27.07	31.95	29.66	30.67	29.19	29.28	26.56	26.60	
	SSIM	8.60	7.57	7.95	7.63	6.96	8.56	7.79	8.16	7.83	7.86	6.93	6.95	
	Time	29.46	2904.16	72962.49	200.57	1652.69	89821.74	217.89	554.87	47.18	1513.44	13.39	86.46	
Div2K ‘Tiger’	PSNR	29.23	26.70	27.04	26.35	25.93	29.56	27.14	28.29	28.03	28.06	26.35	26.36	
	SSIM	7.81	6.64	6.83	5.77	6.60	7.92	6.92	7.22	7.41	7.42	6.85	6.88	
	Time	29.28	2773.43	73193.76	201.57	257.96	90042.40	217.11	502.03	38.45	1334.39	12.53	64.64	
100	PSNR	26.43	24.81	24.51	22.55	25.14	27.21	23.82	25.31	26.07	26.03	25.43	25.40	
	SSIM	6.83	5.92	5.78	3.37	6.44	7.14	5.69	5.57	6.71	6.72	6.55	6.59	
	Time	30.07	2235.74	73188.33	202.24	255.92	89817.23	102.39	505.76	38.48	1187.01	12.60	59.22	

Table B6: Accuracy and runtime of several zero-shot denoising algorithms, measured by PSNR, SSIM and average runtime per image in seconds. Errors denote standard deviation. The best result within each row among the blind zero-shot algorithms are marked bold, while overall best results (including the non-blind algorithms BM3D and DivN) are underlined.

Data set	σ		v-MFA			
			$H = 3$	$H = 5$	$H = 10$	$H = 20$
Set12	25	PSNR	28.71 ± 1.64	29.15 ± 1.40	29.11 ± 1.14	27.61 ± 1.81
		SSIM	8.22 ± 0.38	8.25 ± 0.33	8.05 ± 0.32	7.29 ± 0.68
		Time	1.20 ± 0.81	1.28 ± 0.91	1.47 ± 0.93	2.68 ± 1.49
	50	PSNR	26.19 ± 1.29	26.12 ± 1.19	25.26 ± 1.44	23.16 ± 2.30
		SSIM	7.26 ± 0.38	7.03 ± 0.39	6.30 ± 0.68	5.08 ± 1.07
		Time	1.19 ± 0.67	1.43 ± 0.92	1.62 ± 0.91	3.04 ± 1.54
BSD68	25	PSNR	26.78 ± 3.40	27.25 ± 3.27	27.77 ± 2.86	27.64 ± 2.01
		SSIM	7.30 ± 1.06	7.47 ± 0.94	7.62 ± 0.70	7.43 ± 0.48
		Time	1.22 ± 0.26	1.36 ± 0.25	1.63 ± 0.34	2.77 ± 0.47
	50	PSNR	25.18 ± 2.82	25.30 ± 2.66	25.12 ± 2.19	24.02 ± 1.46
		SSIM	6.54 ± 1.00	6.51 ± 0.86	6.20 ± 0.60	5.34 ± 0.57
		Time	1.28 ± 0.22	1.48 ± 0.31	1.80 ± 0.34	3.20 ± 0.56
Confocal	–	PSNR	36.74 ± 3.60	36.83 ± 3.54	36.28 ± 3.27	34.87 ± 3.13
		SSIM	9.31 ± 0.40	9.34 ± 0.37	9.34 ± 0.36	9.22 ± 0.40
		Time	4.61 ± 2.18	5.39 ± 3.10	5.66 ± 2.35	9.59 ± 3.98
Div2K ‘Tiger’	25	PSNR	28.42	29.19	30.47	31.60
		SSIM	7.57	7.83	8.21	8.49
		Time	43.73	46.48	60.49	89.29
	50	PSNR	27.55	28.03	28.65	29.13
		SSIM	7.24	7.41	7.61	7.76
		Time	34.97	38.65	47.82	74.24

Table B7: The effect of different values of H on v-MFA for denoising. Accuracy and runtime are measured by PSNR, SSIM and average runtime per image in seconds. Errors denote standard deviation. The best results within each row are marked bold.

References

- E. Agustsson and R. Timofte. Ntire 2017 challenge on single image super-resolution: dataset and study. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.
- J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, et al. PyTorch 2: faster machine learning through dynamic python bytecode transformation and graph compilation. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 2, 2024.
- C. Archambeau, N. Delannay, and M. Verleysen. Mixtures of robust probabilistic principal component analyzers. *Neurocomputing*, 71(7):1274–1282, 2008.
- D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.
- H. Asheri, R. Hosseini, and B. N. Araabi. A new EM algorithm for flexibly tied GMMs with large number of components. *Pattern Recognition*, 114:107836, 2021.
- O. Bachem, M. Lucic, H. Hassani, and A. Krause. Fast and provably good seedings for k-means. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 55–63, 2016a.
- O. Bachem, M. Lucic, S. H. Hassani, and A. Krause. Approximate k-means++ in sublinear time. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 30, pages 1459–1467, 2016b.
- A. Banerjee, S. Merugu, I. S. Dhillon, J. Ghosh, and J. Lafferty. Clustering with Bregman Divergences. *Journal of Machine Learning Research*, 6(10):1705–1749, 2005.
- J. Batson and L. Royer. Noise2Self: blind denoising by self-supervision. In *International Conference on Machine Learning, (ICML)*, pages 524–533, 2019a.
- J. Batson and L. Royer. Noise2Self: blind denoising by self-supervision. *GitHub Repository*. <https://github.com/czbiohub-sf/noise2self>, 2019b.
- C.-D. Bei and R. Gray. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, 33(10):1132–1133, 1985.
- H. M. Bennett, W. Stephenson, C. M. Rose, and S. Darmanis. Single-cell proteomics enabled by next-generation sequencing or mass spectrometry. *Nature Methods*, 20(3):363–374, 2023.
- C. M. Bishop. *Pattern Recognition and Machine Learning*. 2006.
- D. M. Blei and M. I. Jordan. Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1(1), 2006.
- M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

- N. Bouguila and W. Fan, editors. *Mixture Models and Applications*. 2020.
- C. Bouveyron, S. Girard, and C. Schmid. High-dimensional data clustering. *Computational Statistics & Data Analysis*, 52(1):502–519, 2007.
- M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *European Conference on Computer Vision (ECCV)*, 2018.
- J. Y. K. Chan and A. P. Leung. Efficient k-means++ with random projection. In *International Joint Conference on Neural Networks (IJCNN)*, pages 94–100, 2017.
- D.-Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham. Fast search algorithms for vector quantization and pattern matching. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 9, pages 372–375, 1984.
- G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. EMNIST: Extending MNIST to handwritten letters. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.
- M. C. Conroy, B. Lacey, J. Bešević, W. Omiyale, Q. Feng, M. Effingham, J. Sellers, S. Sheard, M. Pancholi, G. Gregory, et al. UK Biobank: a globally important resource for cancer research. *British Journal of Cancer*, 128(4):519–527, 2023.
- K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.
- C. Daskalakis, C. Tzamos, and M. Zampetakis. Ten steps of EM suffice for mixtures of two Gaussians. In *Conference on Learning Theory (COLT)*, volume 65, pages 704–710, 2017.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977.
- I. Diakonikolas, D. M. Kane, and A. Stewart. List-decodable robust mean estimation and learning mixtures of spherical Gaussians. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1047–1060, 2018.
- C. Ding and X. He. K-means clustering via principal component analysis. In *International Conference on Machine Learning (ICML)*, page 29, 2004.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, et al. An image is worth 16x16 words: transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.
- J. Drefs, E. Guiraud, and J. Lücke. Evolutionary Variational Optimization of Generative Models. *Journal of Machine Learning Research*, 23(21):1–51, 2022.
- G. Elidan and N. Friedman. The information bottleneck em algorithm. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 200–208, 2002.

- C. Elkan. Using the Triangle Inequality to Accelerate k-Means. In *International Conference on Machine Learning (ICML)*, pages 147–153, 2003.
- M. D. Escobar and M. West. Bayesian Density Estimation and Inference Using Mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.
- G. Exarchakis, O. Oubari, and G. Lenz. A sampling-based approach for efficient clustering in large datasets. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12403–12412, 2022.
- Y. Fang, W. Wang, B. Xie, Q. Sun, L. Wu, et al. EVA: exploring the limits of masked visual representation learning at scale. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19358–19369, 2023.
- W. Fedus, B. Zoph, and N. Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.
- D. Feldman, M. Faulkner, and A. Krause. Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 24, pages 2142–2150, 2011.
- D. Forster, A.-S. Sheikh, and J. Lücke. Neural Simpletrons: Learning in the Limit of Few Labels with Directed Generative Networks. *Neural Computation*, 30(8):2113–2174, 2018.
- N. C. Frey, R. Soklaski, S. Axelrod, S. Samsi, R. Gómez-Bombarelli, et al. Neural scaling of deep chemical models. *Nature Machine Intelligence*, 5(11):1297–1305, 2023.
- P. Fränti and S. Sieranoja. How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, 93:95–112, 2019.
- Z. Ghahramani and G. E. Hinton. The EM Algorithm for Mixtures of Factor Analyzers. Technical Report CRG-TR-96-1, University of Toronto, 1996.
- G. Guennebaud, B. Jacob, et al. Eigen v3. <http://eigen.tuxfamily.org>, 2010.
- S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.
- T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, et al. Scaling Laws for Autoregressive Generative Modeling. *arXiv:2010.14701*, 2020.
- J. Hertrich, D.-P.-L. Nguyen, J.-F. Aujol, D. Bernard, Y. Berthoumieu, et al. PCA Reduced Gaussian Mixture Models with Applications in Superresolution. *Inverse Problems and Imaging*, 16(2):341–366, 2022.
- F. Hirschberger, D. Forster, and J. Lücke. A Variational EM Acceleration for Efficient Clustering at Very Large Scales. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):9787–9801, 2022.

- W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, et al. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.
- A. Hyvärinen. A unified probabilistic model for independent and principal component analysis. In *Advances in Independent Component Analysis and Learning Machines*, pages 75–82. 2015.
- R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991.
- W. Jakob, J. Rhineland, and D. Moldovan. pybind11 – Seamless operability between C++11 and Python. <https://github.com/pybind/pybind11>, 2017.
- M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, 1999.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, et al. Scaling Laws for Neural Language Models. *arXiv:2001.08361*, 2020.
- M. E. Khan, G. Bouchard, K. P. Murphy, and B. M. Marlin. Variational bounds for mixed-data factor analysis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 23, 2010.
- D. Kingma and M. Welling. Efficient gradient-based inference through transformations between Bayes nets and neural nets. In *International Conference on Machine Learning (ICML)*, volume 32, pages 1782–1790, 2014.
- L. Kock, N. Klein, and D. J. Nott. Variational inference and sparsity in high-dimensional deep Gaussian mixture models. *Statistics and Computing*, 32(5):70, 2022.
- A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.
- A. Krull, T.-O. Buchholz, and F. Jug. Noise2Void - learning denoising from single noisy images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2124–2132, 2019a.
- A. Krull, T.-O. Buchholz, and F. Jug. Noise2Void - learning denoising from single noisy images. *GitHub Repository*. <https://github.com/juglab/n2v>, 2019b.
- J. Kwon and C. Caramanis. The EM algorithm gives sample-optimality for learning mixtures of well-separated Gaussians. In *Conference on Learning Theory (COLT)*, volume 125, pages 2425–2487, 2020.
- R. F. Laine, G. Jacquemet, and A. Krull. Imaging in focus: An introduction to denoising bioimages in the era of deep learning. *The International Journal of Biochemistry & Cell Biology*, 140:106077, 2021.
- J. Lequyer, R. Philip, A. Sharma, W.-H. Hsu, and L. Pelletier. A fast blind zero-shot denoiser. *Nature Machine Intelligence*, 4(11):953–963, 2022a.

- J. Lequyer, R. Philip, A. Sharma, W.-H. Hsu, and L. Pelletier. Noise2Fast. *GitHub Repository*. <https://github.com/pelletierlab/Noise2Fast>, 2022b.
- J. Li and A. Barron. Mixture density estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12, 1999.
- W. Lin, M. E. Khan, and M. Schmidt. Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. In *International Conference on Machine Learning (ICML)*, volume 97, pages 3992–4002, 2019.
- A. Liu and J. Li. Clustering mixtures with almost optimal separation in polynomial time. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1248–1261, 2022.
- J. Liu, R. Jiang, X. Liu, F. Zhou, Y. Chen, et al. Large-Scale Clustering on 100M-Scale Datasets Using a Single T4 GPU via Recall KNN and Subgraph Segmentation. *Neural Processing Letters*, 56(1), 2024.
- Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.
- Z. Liu, L. Yu, J. H. Hsiao, and A. B. Chan. PRIMAL-GMM: PaRametrIc MANifold Learning of Gaussian Mixture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):3197–3211, 2022.
- M. Lucic, M. Faulkner, A. Krause, and D. Feldman. Training Gaussian Mixture Models at Scale via Coresets. *Journal of Machine Learning Research*, 18(160):1–25, 2018.
- J. Lücke. Truncated Variational Expectation Maximization. *arXiv:1610.03113v3*, 2019.
- J. Lücke and J. Eggert. Expectation Truncation And the Benefits of Preselection in Training Generative Models. *Journal of Machine Learning Research*, 11(96):2855–2900, 2010.
- Q. Ma, J. Jiang, X. Zhou, P. Liang, X. Liu, and J. Ma. Pixel2Pixel: A Pixelwise Approach for Zero-Shot Single Image Denoising. *GitHub Repository*. <https://github.com/qingma2016/Pixel2Pixel>, 2025a.
- Q. Ma, J. Jiang, X. Zhou, P. Liang, X. Liu, and J. Ma. Pixel2Pixel: A Pixelwise Approach for Zero-Shot Single Image Denoising. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(6):4614–4629, 2025b.
- Y. Mäkinen. Python wrapper for BM3D denoising. *Python Package Index*. <https://pypi.org/project/bm3d>, 2024.
- D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 416–423, 2001.
- V. Maz’ya and G. Schmidt. On approximate approximations using Gaussian kernels. *IMA Journal of Numerical Analysis*, 16(1):13–29, 1996.

- G. McLachlan and D. Peel. *Finite Mixture Models*. 2000.
- G. McLachlan, D. Peel, and R. Bean. Modelling high-dimensional data by mixtures of factor analyzers. *Computational Statistics & Data Analysis*, 41(3):379–388, 2003.
- T. Monnier, T. Groueix, and M. Aubry. Deep transformation-invariant clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 7945–7955, 2020.
- H. Mousavi and J. Lücke. Linear and Nonlinear Generative Models for ‘Zero-Shot’ Image Denoising in the Limit of Few Photons. *Journal of Mathematical Imaging and Vision*, 67(3):1–17, 2025.
- N. Nasios and A. Bors. Variational learning for Gaussian mixture models. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 36(4):849–862, 2006.
- R. Neal and G. Hinton. A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants. *Learning in Graphical Models*, pages 355–368, 1998.
- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, et al. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- H. D. Nguyen, F. Forbes, and G. J. McLachlan. Mini-batch learning of exponential family finite mixture models. *Statistics and Computing*, 30(4):731–748, 2020.
- M.-K. Nguyen-Nhat, R. S. Teo, L. Abdullaev, M. Mok, V.-H. Tran, and T. M. Nguyen. Modeling Expert Interactions in Sparse Mixture of Experts via Graph Structures. *arXiv:2510.16411*, 2025.
- M. Opper and C. Archambeau. The Variational Gaussian Approximation Revisited. *Neural Computation*, 21(3):786–792, 2009.
- E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- M. Prakash, A. Krull, and F. Jug. DivNoising: diversity denoising with fully convolutional variational autoencoders. *GitHub Repository*. <https://github.com/juglab/DivNoising>, 2021a.
- M. Prakash, A. Krull, and F. Jug. Fully unsupervised diversity denoising with convolutional variational autoencoders. In *International Conference on Learning Representations (ICLR)*, 2021b.
- Y. Quan, M. Chen, T. Pang, and H. Ji. Self2Self with dropout: learning self-supervised denoising from single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1887–1895, 2020a.

- Y. Quan, M. Chen, T. Pang, and H. Ji. Self2Self with dropout: learning self-supervised denoising from single image. *GitHub Repository*. <https://github.com/scut-mingqinchen/self2self>, 2020b.
- L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- O. Regev and A. Vijayaraghavan. On learning mixtures of well-separated Gaussians. In *Symposium on Foundations of Computer Science (FOCS)*, pages 85–96, 2017.
- E. Richardson. torch-mfa. *GitHub Repository*. <https://github.com/eitanrich/torch-mfa>, 2019.
- E. Richardson and Y. Weiss. On GANs and GMMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.
- S. Robin and L. Scrucca. Mixture-based estimation of entropy. *Computational Statistics & Data Analysis*, 177:107582, 2023.
- S. Salwig, J. Drefs, and J. Lücke. Zero-shot denoising of microscopy images recorded at high-resolution limits. *PLOS Computational Biology*, 20(6):e1012192, 2024.
- D. Sculley. Web-scale k-means clustering. In *International Conference on World Wide Web (WWW)*, pages 1177–1178, 2010.
- N. Segol and B. Nadler. Improved convergence guarantees for learning Gaussian mixture models by EM and gradient EM. *Electronic Journal of Statistics*, 15(2), 2021.
- N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations (ICLR)*, 2017.
- J. A. Shelton, J. Gasthaus, Z. Dai, J. Lücke, and A. Gretton. GP-Select: Accelerating EM Using Adaptive Subspace Preselection. *Neural Computation*, 29(8):2177–2202, 2017.
- A. Shocher, N. Cohen, and M. Irani. Zero-shot super-resolution using deep internal learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3118–3126, 2018.
- O. Tange. GNU Parallel 20210822 ('Kabul'), 2021.
- The Boost organization. Boost C++ Libraries. <http://www.boost.org>, 2024.
- The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437(7063):1299–1320, 2005.
- L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2016.
- B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, et al. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

- T. Tian, J. Wan, Q. Song, and Z. Wei. Clustering single-cell RNA-seq data with a model-based deep learning approach. *Nature Machine Intelligence*, 1(4):191–198, 2019.
- D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. *GitHub Repository*. <https://github.com/DmitryUlyanov/deep-image-prior>, 2018a.
- D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9446–9454, 2018b.
- M. Wortsman, T. Dettmers, L. Zettlemoyer, A. Morcos, A. Farhadi, and L. Schmidt. Stable and low-precision training for large-scale vision-language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 10271–10298, 2023.
- J. Xu, D. J. Hsu, and A. Maleki. Global analysis of Expectation Maximization for mixtures of two Gaussians. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.
- L. Xu, M. Jordan, and G. E. Hinton. An Alternative Model for Mixtures of Experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 7, 1994.
- W. Xu, M. Fazel, and S. Du. Toward global convergence of gradient EM for over-paramterized Gaussian mixture models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10770–10800, 2024.
- A. J. Zeevi and R. Meir. Density Estimation Through Convex Combinations of Densities: Approximation and Estimation Bounds. *Neural Networks*, 10(1):99–109, 1997.
- Y. Zhang, Y. Zhu, E. Nichols, Q. Wang, S. Zhang, C. Smith, and S. Howard. A Poisson-Gaussian denoising dataset with real fluorescence microscopy images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11710–11718, 2019.
- J.-H. Zhao and P. L. H. Yu. Fast ML Estimation for the Mixture of Factor Analyzers via an ECM Algorithm. *IEEE Transactions on Neural Networks*, 19(11):1956–1961, 2008.
- M. Zhou and L. Carin. Negative Binomial Process Count and Mixture Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):307–320, 2015.
- D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *International Conference on Computer Vision (ICCV)*, pages 479–486, 2011.