Variational Optimization for Scalable Unsupervised and Semi-supervised Probabilistic Learning

by

Sebastian Salwig

This thesis has been accepted in fulfillment

of the requirements for the degree of

Doctor of Natural Sciences

in Faculty V - Mathematics and Science of

Carl von Ossietzky University of Oldenburg

2025

Variational Optimization for Scalable Unsupervised and Semi-supervised Probabilistic Learning

Doctoral Thesis by Sebastian Salwig

Born in Oldenburg (Oldb), Germany

on January 25$^{\text{th}}$, 1996

Promotion Committee

Chair:	Prof. Dr. Jörg Lücke, University of Innsbruck, Austria
2$^{\text{nd}}$ Reviewer:	Assoz. Prof. Antonio Rodríguez-Sánchez, PhD., University of Innsbruck, Austria
3$^{\text{rd}}$ Reviewer:	Prof. Dr. Alexander K. Hartmann, Carl von Ossietzky University of Oldenburg, Germany
4$^{\text{th}}$ Reviewer:	Lecturer Dr. Georgios Exarchakis, University of Bath, United Kingdom

Date of Defense

November 14, 2025

Abstract

With the growing availability of data and the frequent scarcity of labels, semi-supervised and unsupervised learning algorithms have become increasingly important in the field of machine learning. However, scaling these algorithms to large, high-dimensional datasets often incurs substantial computational costs. In this thesis, we develop, implement and investigate highly efficient algorithms for semi-supervised and unsupervised learning. A central focus is the efficient optimization of probabilistic generative models, which describe a data-generating process from latent to observed variables and are typically trained by approximating maximum likelihood learning given the observed data under the model. Concretely, we investigate and derive efficient optimization algorithms based on truncated variational posterior approximations, which are applicable to probabilistic generative models with discrete (in particular, categorical) latent variables. By integrating the derived variational optimization technique with mixtures of factor analyzers (MFAs), we propose a highly efficient learning algorithm that exhibits sublinear scaling with respect to both model size and data size. Remarkably, the resulting v-MFA algorithm enables the novel training of large-scale Gaussian mixture models (GMMs) at scales previously not reachable using standard hardware. Another well-established class of approaches in semi-supervised and unsupervised learning relies on graph-based optimization, which encodes intrinsic data structure through graph representations. We focus on a class of graph-based semi-supervised learning (GSSL) methods that leverage data distribution models to reduce the computational burden associated with large-scale conventional GSSL. While these methods significantly lower the costs of graph construction and label propagation, optimizing the underlying data distribution models can itself become a new computational bottleneck. By incorporating our variational optimization techniques into a recent distribution-based GSSL framework, we propose a novel GSSL approach that overcomes this bottleneck. As distribution models, we use GMMs with diagonal covariance matrices as well as MFAs, denoted as v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL, respectively. The resulting algorithms establish new state-of-the-art results on standard GSSL benchmarks (with v-MFA-GL outperforming existing methods by a large margin on several benchmarks). Beyond semi-supervised classification, we also evaluate elementary generative models optimized with variational techniques in the task of blind zero-shot denoising. In this setting, an algorithm receives only a single noisy image without access to additional training data, ground truth or noise-level information. Our evaluation datasets comprise both macroscopic natural images and microscopy images, including images of SARS-CoV-2 viruses. Compared to advanced filtering techniques and state-of-the-art deep neural network methods, our algorithms achieve highly competitive denoising performance under blind zero-shot conditions. In particular, the proposed v-MFA algorithm demonstrates strong denoising performance while maintaining very short runtimes.

Zusammenfassung

Mit der zunehmenden Verfügbarkeit von Daten und der häufigen Knappheit von Labels haben halbüberwachte (*engl.* semi-supervised) und unüberwachte (*engl.* unsupervised) Lernalgorithmen im Bereich des maschinellen Lernens zunehmend an Bedeutung gewonnen. Die Skalierung dieser Algorithmen auf große, hochdimensionale Datensätze ist jedoch oft mit erheblichen Rechenkosten verbunden. In dieser Arbeit entwickeln, implementieren und untersuchen wir hocheffiziente Algorithmen für halbüberwachtes und unüberwachtes Lernen. Ein zentraler Schwerpunkt ist die effiziente Optimierung probabilistischer generativer Modelle, die einen datengenerierenden Prozess von latenten zu beobachteten Variablen beschreiben und typischerweise durch Approximation der Maximum-Likelihood-Methode unter Berücksichtigung der beobachteten Daten unter dem Modell trainiert werden. Konkret untersuchen und leiten wir effiziente Optimierungsalgorithmen ab, die auf trunkierten variationalen Approximationen von a-posteriori-Verteilungen basieren und auf probabilistische generative Modelle mit diskreten (insbesondere kategoriale) latenten Variablen anwendbar sind. Durch die Integration der abgeleiteten variationalen Optimierungstechnik mit Mixtures of Factor Analyzers (MFAs) schlagen wir einen hocheffizienten Lernalgorithmus vor, der in Bezug auf die Modellgröße und die Datengröße eine sublineare Skalierung aufweist. Bemerkenswert ist, dass der daraus resultierende v-MFA-Algorithmus das neuartige Training groß angelegter Gaußscher Mischmodelle (*engl.* Gaussian mixture models, GMMs) in einem Umfang ermöglicht, der mit Standardhardware bisher nicht erreichbar war. Eine weitere etablierte Klasse von Ansätzen im halbüberwachten und unüberwachten Lernen basiert auf graphbasierter Optimierung, die die intrinsische Datenstruktur durch Graphdarstellungen kodiert. Wir konzentrieren uns auf eine Klasse von graphbasierten halbüberwachten Lernmethoden (*engl.* graph-based semi-supervised learning, GSSL), die Datenverteilungsmodelle nutzen, um den Rechenaufwand im Zusammenhang mit groß angelegten konventionellen GSSL zu reduzieren. Während diese Methoden die Kosten für die Graphkonstruktion und die Label-Propagierung erheblich senken, kann die Optimierung der zugrunde liegenden Datenverteilungsmodelle selbst zu einem neuen Rechenengpass werden. Durch die Einbindung unserer variationalen Optimierungstechniken in ein aktuelles verteilungsbasiertes GSSL-Framework schlagen wir einen neuartigen GSSL-Ansatz vor, der diesen Engpass überwindet. Als Verteilungsmodelle verwenden wir GMMs mit diagonalen Kovarianzmatrizen sowie MFAs, die jeweils als v-GMM$^{\mathrm{d}}$-GL und v-MFA-GL bezeichnet werden. Die daraus resultierenden Algorithmen erzielen neue State-of-the-Art Ergebnisse bei Standard-GSSL-Benchmarks (wobei v-MFA-GL bei mehreren Benchmarks die bestehenden Methoden deutlich übertrifft). Über die halbüberwachte Klassifizierung hinaus bewerten wir auch elementare generative Modelle, die mit variationalen Techniken für die Aufgabe der blinden Zero-Shot-Rauschunterdrückung optimiert wurden. In dieser Konfiguration erhält ein Algorithmus nur ein einziges verrauschtes Bild, ohne Zugriff auf zusätzliche Trainingsdaten, Ground-Truth oder Informationen zum Rauschpegel. Unsere Bewertungs-

datensätze umfassen sowohl makroskopische als auch mikroskopische Bilder, darunter Bilder von SARS-CoV-2-Viren. Im Vergleich zu fortschrittlichen Filtertechniken und modernsten Methoden mit tiefen neuronalen Netzen erzielen unsere Algorithmen unter blinden Zero-Shot-Bedingungen eine äußerst wettbewerbsfähige Rauschunterdrückungsleistung. Insbesondere der vorgeschlagene v-MFA-Algorithmus zeigt eine starke Rauschunterdrückungsleistung bei gleichzeitig sehr kurzen Laufzeiten.

Declaration of Authorship

I, Sebastian Salwig, declare that this thesis and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this university.
- I am aware of the guidelines of good scientific practice of the Carl von Ossietzky University of Oldenburg and observed them.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- I have not availed myself of any commercial placement or consulting services in connection with my promotion procedure.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date

Sebastian Salwig

Acknowledgments

I am truly thankful to everyone who supported me throughout this thesis.

My special thanks go to Jörg Lücke for his valuable advice, continuous encouragement and the many constructive discussions that motivated and supported me throughout the entire duration of my doctoral studies.

I would also like to thank Jakob Drefs and Till Kahlke for the close and productive collaborations, as well as all other former and current colleagues at the Machine Learning Lab at the University of Oldenburg, with whom I shared many meals and discussions. Thanks for the great time!

Finally, I am deeply grateful to my girlfriend and parents for always being there with their support and encouragement.

Contents

List of Symbols

We adopt the following notation conventions throughout this thesis (if not otherwise stated):

- Lowercase letters (e.g., σ) denote scalars, while bold lowercase letters (e.g., $\boldsymbol{x}$) represent vectors. Bold capital letters (e.g., $\boldsymbol{W}$ or $\boldsymbol{\Sigma}$) are used to denote matrices. Individual vector or matrix elements are denoted using subscript notation, such as x_d or W_{cd}, where the subscripts indicate specific row and column indices. The c-th row (or column) of matrix $\boldsymbol{W}$ is denoted by $\boldsymbol{W}_c$.
- $\boldsymbol{x}$ denotes a vector of observed variables (data point) with dimension D.
- $\boldsymbol{s}$ denotes a vector of binary hidden states of dimensionality H, $\boldsymbol{z}$ is a vector of continuous hidden states of dimensionality H, and c is a discrete mixture component index (i.e., cluster), taking values in $\{1, \ldots, C\}$.
- $\boldsymbol{\mathcal{X}} = \boldsymbol{x}_{1:N}$ denotes N data points drawn independently and identically distributed (i.i.d.) from a probability distribution. The subscript n is used to refer to the n-th element of the set (i.e., $\boldsymbol{x}_n$).
- $\boldsymbol{\mathcal{Y}} = y_{1:N_l}$ denotes a set of labels, where the first N_l data points are labeled, and each label y_n belongs to one of K discrete classes, i.e., $y_n \in \{1, ..., K\}$.
- $\boldsymbol{\Theta}$ denotes the set of model parameters.
- $p(\cdot)$ and p are used to represent a probability density function.
- $q_n(c; \boldsymbol{\Theta})$ denotes the variational distribution defined over the latent space for data point $\boldsymbol{x}_n$, for each $n = 1, \ldots, N$.
- $\mathcal{K}^{(n)}$ denotes a set of finite hidden states c corresponding to the data point $\boldsymbol{x}_n$. We further use $\boldsymbol{\mathcal{K}}$ to refer to the set of all $\mathcal{K}^{(n)}$, i.e., $\boldsymbol{\mathcal{K}} = \{\mathcal{K}^{(1)}, \ldots, \mathcal{K}^{(N)}\}$. The size of each set $\mathcal{K}^{(n)}$ is denoted by C'.
- $\mathcal{S}^{(n)}$ denotes the search space corresponding to the data point $\boldsymbol{x}_n$. The maximal size of each set $\mathcal{S}^{(n)}$ is denoted by S.
- $\mathbb{E}_{p(\cdot)}$ denotes the expectation value w.r.t. the distribution $p(\cdot)$.
- The superscript $\top$ (or T) denotes the transpose of a vector or a matrix.
- $\mathcal{O}(\cdot)$ denotes Big O notation, i.e., asymptotic complexity.

Chapter 1

Introduction

The extraction of information from data and the formulation of well-founded conclusions based on this information have long been essential in driving progress across various fields, including research, education, healthcare and social systems. Data often hold valuable insights that help us to understand complex relationships and gain a clearer understanding of the world around us. For example, fundamental physical laws have been discovered or verified through the systematic collection and analysis of empirical measurements, just as modern physicians diagnose diseases through the evaluation of medical image or using data of other measurements. The rise of digital technologies has greatly expanded the ease and scope of data collection, while the advent of the Internet has created a platform for the global sharing of information. In recent decades, the volume of recorded data has increased exponentially, often to such an extent that humans alone are unable to process or interpret it. However, the sheer amount of data is often not the only factor that poses a challenge in analyzing data. Extracting meaningful information often requires addressing complex patterns, high dimensionality, uncertainties and variabilities. Additionally, data are often affected by noise, incomplete information and may be imbalanced or non-stationary over time. This becomes particularly challenging in domains where data remains difficult to obtain, as data acquisition can still be highly cost-intensive or may rely exclusively on specialized sensors or manual processes.

Machine learning (ML) algorithms have therefore become essential tools for processing and extracting information from data. Unlike rule-based approaches in early AI systems, ML algorithms are data-driven and often capable of solving complex tasks that are difficult or impractical for manually defined rules to handle. These algorithms rely on mathematical models that typically learn internal representations of the data, which can then be applied to solve tasks such as prediction, classification or clustering (and many more). Given that the process of data extraction is often affected by various degrees of uncertainty, probabilistic ML algorithms are of particular relevance and are also central to this thesis. Such methods explicitly model and account for uncertainty in data through the use of probability distributions. To allow for interpretable and robust decision-making, Bayes' rule is a widely used and foundational principle for incorporating uncertainty and prior knowledge, which is especially valuable when data are scarce or decisions are critical.

In recent years, the field of ML has undergone rapid advancements in performance and the versatility of its applications. Today, ML tools are employed in nearly all areas where data-driven decision-making is essential. In medicine, ML models are used to support

disease diagnosis (e.g., Ahsan *et al.*, 2022; Shehab *et al.*, 2022) or automatically analyze medical images and patient data (e.g., Chen *et al.*, 2022), for example, to detect cancer in biopsy images (Araújo *et al.*, 2017). In computer vision and audio signal processing, ML approaches are used for various signal enhancement tasks, including denoising (e.g., Mousavi and Lücke, 2025; Prakash *et al.*, 2021b; Quan *et al.*, 2020a; Zoran and Weiss, 2011), inpainting (e.g., Drefs *et al.*, 2022), and super-resolution (e.g., Shocher *et al.*, 2018). ML-based object and face detection systems (e.g., Redmon *et al.*, 2016) are fundamental to applications such as autonomous driving (Jia *et al.*, 2023) and biometric authentication on mobile devices (Alrawili *et al.*, 2024). Speech recognition (e.g., Baevski *et al.*, 2021; Hsu *et al.*, 2021; Workshop *et al.*, 2022) represents another area of application, supporting technologies like virtual assistants (e.g., Siri and Alexa). Large language models (e.g., Brown *et al.*, 2020; Chowdhery *et al.*, 2023), along with dialogue agents like ChatGPT (Achiam *et al.*, 2023), have enabled advanced text generation systems that can produce coherent, contextually relevant and human-like responses. Also more generally, generative models (e.g., Goodfellow *et al.*, 2014; Ho *et al.*, 2020; Richardson and Weiss, 2018) have shown impressive capabilities in image synthesis, particularly diffusion models in large-scale text-to-image generation (e.g., Rombach *et al.*, 2022; Saharia *et al.*, 2022). And these examples represent only a small subset of the broader range of ML applications.

Remarkably, the 2024 Nobel Prize in Physics was awarded to researchers in the field of artificial intelligence. Geoffrey Hinton and John J. Hopfield received the prize for their foundational contributions to the theoretical underpinnings of modern deep learning (Nobel Prize, 2024). While Hopfield introduced physical concepts such as energy minimization into neural networks (e.g., Hopfield, 1982), Hinton applied probabilistic concepts from statistical physics to develop the Boltzmann machine (Hinton and Sejnowski, 1986).

The success of modern ML algorithms can be attributed to several factors, which together play a crucial role in making these technologies so powerful and widely adopted. One contributing factor has been the exponential growth in the availability of data. "Big data" has become a foundation upon which many of today's ML algorithms are built. Another important factor is the advancement in computational power. The development of specialized hardware components such as multi-core CPUs and especially GPUs has dramatically increased the processing speed of ML algorithms. This hardware enables parallel processing of large datasets and models. Equally important are the advancements in developing powerful models and scalable optimization techniques, along with their integration of mathematical theories from probability and statistics.

One prominent class of ML models that has significantly benefited from these factors are deep learning (DL) models, i.e., models incorporating deep neural networks (DNNs). Concretely, the availability of large-scale datasets, advances in computational resources and the progress in algorithm development have enabled the scaling of neural networks to increasingly larger sizes (i.e., models with a large number of model parameters). Following the success of "AlexNet" (at the time one of the largest convolutional neural networks; Krizhevsky *et al.*, 2012) in the ImageNet competition, the design of sophisticated and efficiently scalable neural network architectures (e.g., He *et al.*, 2016; Ronneberger *et al.*, 2015; Vaswani *et al.*, 2017) became a central focus of ML research. Moreover, continuous improvements of optimization methods such as gradient descent (e.g., Kingma and Ba, 2014) have facilitated the efficient training of these large models on vast datasets. As a result, DL has emerged as a dominant modeling paradigm within the field of ML over the past decade.

The integration of DNNs (with their universal function approximation property; e.g., Cybenko, 1989; Hornik *et al.*, 1989) with probabilistic data models, particularly probabilistic generative models, has substantially driven the field of ML (e.g., Kingma and Welling, 2014; Larochelle and Murray, 2011; Rezende and Mohamed, 2015; Sohl-Dickstein *et al.*, 2015). The resulting deep generative models (DGMs) have attracted widespread attention for their ability to generate new, realistic data such as images, speech or written text. As with DL more broadly, a major contributor to the success of DGMs has been the scaling of neural networks to unprecedented sizes, combined with training on internet-scale datasets, which has enhanced their performance and capabilities (e.g., Henighan *et al.*, 2020; Kaplan *et al.*, 2020). Today, the largest of these models contain parameter counts on the order of hundreds of billions (e.g., Brown *et al.*, 2020; Chowdhery *et al.*, 2023).

The training and inference of these large-scale models, however, have been accompanied by a substantial increase in computational costs, leading to exponentially higher energy demands and carbon emissions. Recent studies indicate that the cost of training the most computationally intensive models has been escalating rapidly, growing at an average rate of 2.4× annually since 2016 (Cottier *et al.*, 2024). For example, the training of GPT-4 is estimated to have required computational resources valued at approximately $78 million (Maslej *et al.*, 2024). Accordingly, the training cost of a frontier AI model in 2027 is projected to exceed $1 billion. This exponential increase in resource requirements has led to a concentration of ML training capabilities for such large models within a small number of well-resourced entities, primarily large technology companies. Such concentration may limit the diversity of perspectives and research approaches, particularly from academia and the wider society. In addition to their high computational costs, DL models exhibit several other inherent limitations including limited interpretability, challenges in adaptability and usability, dependence on careful hyperparameter tuning and often a strong reliance on large volumes of labeled data. Their theoretical understanding lags behind their empirical success, and they are often considered as black-box systems, with internally learned representations that are not interpretable by humans (for discussions see, e.g., Manduchi *et al.*, 2024).

This progress highlights two important insights for the field of ML: (i) recent advances in generative modeling have been primarily driven by scaling DGMs, i.e., increasing the number of model parameters of the underlying DNNs; and (ii) there remains a continuing need for developing scalable, complementary approaches.

In this thesis, we investigate highly efficient optimization methods for training large-scale models (e.g., models with millions or billions of parameters) in semi-supervised and unsupervised learning. In deliberate contrast to other such large-scale ML models, the scalability of the approaches here studied will not be inherited from the scalability of DNNs. Like other DNN-based generative models, we focus on probabilistic data models, which define an explicit data-generating process and incorporate latent variables to generate the observed data. Such models are ideally optimized by maximizing the likelihood of the observed data under the assumed model. However, in real-world scenarios, data are often generated by numerous hidden factors and explicitly modeling even a subset of these latent variables can result in highly (computationally) expensive or intractable optimization problems.

To overcome this limitation, the optimization problem is often reformulated as the maximization of a tractable lower bound of the exact likelihood – a foundational concept in variational optimization techniques (e.g., Jordan *et al.*, 1999; Kingma and Welling, 2014; Neal and Hinton, 1998; Opper *et al.*, 2005). A well-known example of such a variational

approach is the variational Expectation Maximization (EM) algorithm (Neal and Hinton, 1998), which is commonly applied to estimate the parameters of latent variable models. In conventional EM (Dempster *et al.*, 1977), the E-step involves computing the posterior distribution over latent variables, which is often intractable or computationally expensive. Variational EM approximates the posterior using variational distributions, which are tractable, parameterized distributions, often from a simpler family than the true posterior (also compare Sec. 1.1).

We focus here on variational optimization techniques that utilize truncated posterior distributions (Lücke, 2019) to enable scalable optimization of discrete latent variable models (additional approximation schemes are discussed in Sec. 1.1.1). The central assumption underlying truncated posteriors is that most of the probability mass is concentrated within a relatively small subset of latent states, while the probability mass of the remaining states is explicitly set to zero (compare Sec. 1.1.1). This assumption has been empirically observed to be well suited for most natural data (cf. Hughes and Sudderth, 2016; Lücke and Eggert, 2010; Shelton *et al.*, 2017). Furthermore, the sparsity in the posterior structure has enabled the development of efficient algorithms for parameter optimization in probabilistic data models, particularly when dealing with large or combinatorially complex latent spaces. The concept of truncated posteriors was first introduced by Lücke and Sahani (2008) and Lücke and Eggert (2010), and has since been applied to a range of both elementary and more sophisticated generative models (e.g., Dai *et al.*, 2013; Dai and Lücke, 2014; Hughes and Sudderth, 2016; Sheikh *et al.*, 2019, 2014; Shelton *et al.*, 2011, 2017). While these early approaches to truncated posteriors relied on (amortized) selection functions, more recent work has introduced fully variational formulations of truncated posteriors (e.g., Drefs *et al.*, 2022, 2023; Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Lücke, 2019; Mousavi *et al.*, 2023). Notably, the application of fully variational optimization using truncated posteriors to Gaussian mixture models (GMMs) has enabled the optimization of the largest-scale mixture models at that time (see Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022). However, this variational EM algorithm was developed for and applied only to GMMs with isotropic and diagonal covariances. Ideally, such optimization methods should be applicable to a broader range of models used in semi-supervised and unsupervised learning. Furthermore, constraining GMMs to isotropic or diagonal covariances hinders the modeling of intra-component correlations which can be crucial for capturing complex data structures.

In this thesis, we develop a variational approach that can be applied to GMMs with arbitrary covariance matrices. We base our derivations on earlier work (primarily Hirschberger *et al.*, 2022; Lücke, 2019; Lücke and Eggert, 2010; Shelton *et al.*, 2017). Very centrally for large parts of this thesis, we can then apply, for the first time, highly efficient variational optimization to mixtures of factor analyzers (MFAs; Ghahramani and Hinton, 1996; McLachlan *et al.*, 2003; Richardson and Weiss, 2018). MFAs can be regarded as a special form of GMMs with low-rank (plus diagonal) covariance matrices. The resulting learning algorithm, denoted as v-MFA, exhibits sublinear scaling w.r.t. model size and data size and allows for the training of large-scale representations on the order of billions of parameters – a scale that was previously attainable only by DNN-based approaches.

A salient hallmark of probabilistic generative models is their ability to learn a data density model of the data generating process. Leveraging the learned data densities, they can address a wide range of downstream tasks, including classification, image enhancement, outlier detection, data generation, compression and many more. In this context, GMMs

play a central role due to their ability to flexibly approximate complex data densities with increasing size (e.g., Li and Barron, 1999; Maz'ya and Schmidt, 1996; Zeevi and Meir, 1997). In theory, any task that requires a sufficiently accurate parametric data density model in $\mathbb{R}^D$ can also be tackled with a sufficiently large and well-optimized GMM (where D denotes the data dimensionality).

The downstream tasks considered in this work are: denoising and semi-supervised classification. Denoising refers to the process of removing noise from data, e.g., images or audio recordings, to recover the underlying clean signal. Specifically, we focus on blind zero-shot denoising (cf. Boukun *et al.*, 2024; Lequyer *et al.*, 2022b; Mousavi and Lücke, 2025; Quan *et al.*, 2020a), where the algorithm is provided only with the noisy data, without access to clean reference data or additional training data. This scenario is particularly relevant in domains, such as astronomy, microscopy and medical imaging, where data acquisition can be prohibitively expensive and high-quality, well-annotated or ground-truth data may be extremely limited or entirely unavailable. In order to enable timely and critical decision-making, the learning algorithm has not only to achieve optimal denoising performance but also support highly efficient training and inference.

Semi-supervised classification involves training a model using a combination of a small set of labeled data and a larger set of unlabeled data in order to improve classification performance by leveraging the structure of the unlabeled data (compare e.g., van Engelen and Hoos, 2020). In order to apply probabilistic generative models to the task of semi-supervised classification, we adopt graph-based semi-supervised learning (GSSL) approaches that exploits the underlying structure of the data by representing it as a graph. In conventional GSSL methods, data points are typically modeled as nodes in the graph and edges between these nodes are constructed based on a similarity measure. Label information is then propagated from labeled nodes to unlabeled nodes over the graph until a global state is reached (Zhu and Ghahramani, 2002). The optimization in label propagation can often be formulated as the minimization of a cost function, for which a closed-form solution can be derived (e.g., Zhou *et al.*, 2003; Zhu *et al.*, 2003). However, computing this closed-form solution typically incurs cubic time complexity $\mathcal{O}(N^3)$ w.r.t. the number of data points N and limits the scalability of label propagation algorithms to large-scale datasets.

One promising line of research seeks to mitigate this computational burden by first learning a data distribution model, followed by graph construction and label propagation on the learned representations (e.g., Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023). Nevertheless, optimizing the distribution model can itself become computationally intensive on large datasets, often emerging as a bottleneck in these approaches. In this thesis, we combine our novel variational optimization techniques with an established GSSL framework (Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023) and propose a highly efficient learning algorithm for semi-supervised learning that addresses this bottleneck.

A significant aspect of this thesis involves not only the theoretical derivation of the algorithms but also their practical implementation and parallelization. This is particularly relevant in fields such as ML, where algorithmic efficiency and practical applicability are essential. To this end, special emphasis was placed on developing software implementations of the proposed algorithms that efficiently utilize the available hardware and leverage multi-core CPU architectures. To facilitate adoption and reproducibility, we have developed lightweight and user-friendly Python frameworks (with core functionalities implemented in C++) that enable other research groups to readily apply the novel methods in their own work.

1.1 Efficient Optimization of Probabilistic Generative Models

In this section, we discuss how probabilistic generative models can be efficiently optimized. We focus on latent variable generative models, which define a data-generating process from latent (hidden) variables to observable data. The structure of these models is typically represented using a directed acyclic graph, also known as a Bayesian network (or Bayes net) encoding the conditional dependencies among variables. A wide range of latent variable generative models that can be expressed as directed acyclic graphs has been proposed in the literature. Prominent examples include mixture models (McLachlan and Peel, 2000), probabilistic principal component analysis (p-PCA; Tipping and Bishop, 1999), factor analysis (e.g., Bishop, 2006), sparse coding models (e.g., Drefs *et al.*, 2022; Olshausen and Field, 1996), hidden Markov models (HMMs; e.g., Murphy, 2012) and deep generative models such as variational autoencoders (VAEs; Drefs *et al.*, 2023; Kingma and Ba, 2014; Rezende *et al.*, 2014), diffusion models (Ho *et al.*, 2020; Sohl-Dickstein *et al.*, 2015) and deep Gaussian mixture models (e.g., Viroli and McLachlan, 2019).

In this thesis, we primarily focus on latent variable models in which the latent variables $\boldsymbol{s}$ are discrete and binary-valued (i.e., each component of $\boldsymbol{s}$ takes a value in $\{0, 1\}$), and the observed variables $\boldsymbol{x}_n$ are continuous or discrete. The generative model defines a joint distribution over $\boldsymbol{x}_n$ and $\boldsymbol{s}$, parameterized by $\boldsymbol{\Theta}$, such that the marginal likelihood of the observed data is given by

$$p(\boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) = \sum_{\boldsymbol{s}} p(\boldsymbol{x}_n, \boldsymbol{s} \,|\, \boldsymbol{\Theta}), \tag{1.1}$$

where the summation is taken over all configurations of the binary latent vector $\boldsymbol{s}$.

Mixture models are a special case of binary latent variable models where the latent vector $\boldsymbol{s}$ is one-hot encoded, i.e., exactly one element equals one while all others are zero. In the context of mixture models, it is conventional to denote the latent variable by a discrete scalar $c \in \{1, \ldots, C\}$, representing one of C possible hidden variables (or components/clusters). Since this thesis primarily focuses on mixture models, we adopt this scalar notation in the following:

$$p(\boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) = \sum_{c} p(\boldsymbol{x}_n, c \,|\, \boldsymbol{\Theta}). \tag{1.2}$$

In order to fit the generative model to data, we aim to find parameter values $\boldsymbol{\Theta}$ that maximize the log-likelihood of the observed data under the model. Given a set of N independently distributed data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the log-likelihood is defined as

$$\mathcal{L}(\boldsymbol{x}_{1:N}; \boldsymbol{\Theta}) = \sum_{n} \log p(\boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) = \sum_{n} \log\Big(\sum_{c} p(\boldsymbol{x}_n, c \,|\, \boldsymbol{\Theta}) \Big). \tag{1.3}$$

As direct maximization of the log-likelihood is often computationally intractable due to the summation over latent variables, the Expectation-Maximization (EM) algorithm is a standard approach for parameter estimation in latent variable models (Dempster *et al.*, 1977). In its conventional formulation, the EM algorithm alternates between the E-step, which computes the posterior distribution $p(c \,|\, \boldsymbol{x}_n, \boldsymbol{\Theta})$ for each data point $\boldsymbol{x}_n$ and latent state c; and the M-step, which updates the model parameters $\boldsymbol{\Theta}$. Repeated iteration of the E- and M-steps guarantees a monotonic increase in the log-likelihood, converging

to a stationary point of the log-likelihood (which may be a local maximum or saddle point).

For models with relatively elementary generative formulations, e.g., Poisson mixture models (PMMs; compare Eq. (1.8)), and moderately sized latent spaces, conventional EM often provides an efficient and tractable method for parameter estimation. However, for more expressive generative models or those with large and combinatorially complex latent spaces, the optimization with conventional EM can quickly become computationally prohibitive or intractable, necessitating approximations.

1.1.1 Variational Optimization

Numerous approximation strategies have been developed to overcome the limitations of the conventional EM algorithm, which arise from the intractability or high computational costs of computing the posterior distribution in the E-step. These approximations span a broad spectrum, including, e.g., sampling-based methods for estimating expectation values w.r.t. the posterior distribution (e.g., Mohamed *et al.*, 2012; Zhou *et al.*, 2009), maximum a posteriori (MAP)/ "hard EM" variants that replace expectations with point estimates (e.g., Olshausen and Field, 1996; Van den Oord and Schrauwen, 2014) and variational EM approaches (e.g., Jordan *et al.*, 1999; Neal and Hinton, 1998; Opper *et al.*, 2005; Saul and Jordan, 1995). Additional approximation strategies (especially related to mixture model-like approaches) are discussed in Chs. 3 and 4.

In the following, we focus on variational EM approximations, which aim to optimize a tractable lower bound on the log-likelihood, known as the variational free energy (Neal and Hinton, 1998) or the evidence lower bound (ELBO; Hoffman *et al.*, 2013). This variational free energy $\mathcal{F}$ is given by

$$\mathcal{L}(\boldsymbol{x}_{1:N};\boldsymbol{\Theta}) \geq \mathcal{F}(\boldsymbol{x}_{1:N};\tilde{\boldsymbol{\Theta}},\boldsymbol{\Theta}) := \sum_n \sum_c q_n(c;\,\tilde{\boldsymbol{\Theta}}) \log p(\boldsymbol{x}_n, c \,|\, \boldsymbol{\Theta}) + \sum_n \mathcal{H}\left[q_n\right], \tag{1.4}$$

where $q_n(c;\,\tilde{\boldsymbol{\Theta}})$ denote the variational distributions with the variational parameters $\tilde{\boldsymbol{\Theta}}$ and $\mathcal{H}\left[q_n\right] = -\mathbb{E}_{q_n}[\log q_n(c;\,\tilde{\boldsymbol{\Theta}})]$ is the Shannon entropy. By selecting variational distributions that avoid the need for summation (or integration, in the case of continuous hidden variables) over large hidden spaces, the optimization of the variational free energy becomes tractable or more efficient. The design of variational distributions typically involves making assumptions about the structure of the posterior distribution, e.g., assuming Gaussianity (e.g., Kingma and Ba, 2014; Opper *et al.*, 2005) or adopting factorized forms (i.e., *mean-field* approximations; e.g., Jordan *et al.*, 1999). Another class of variational distributions that is particularly well-suited for discrete latent variable models are truncated posteriors (e.g., Drefs *et al.*, 2022; Lücke and Eggert, 2010; Shelton *et al.*, 2017), which assume that the majority of the posterior probability mass is concentrated in a relatively small subset of latent states. The posterior probabilities of the remaining states are explicitly set to zero, and the latent states with non-zero probability are treated as variational parameters (for a comprehensive discussion of the advantages (and limitations) of such approximation schemes, see Lücke (2019)). The truncated posterior distributions are defined as

$$q(c;\boldsymbol{x}_n,\mathcal{K}^{(n)},\tilde{\boldsymbol{\Theta}}) = \frac{p(c,\boldsymbol{x}_n \,|\, \tilde{\boldsymbol{\Theta}})}{\sum_{\tilde{c}\in\mathcal{K}^{(n)}} p(\tilde{c},\boldsymbol{x}_n \,|\, \tilde{\boldsymbol{\Theta}})}\, \delta(c \in \mathcal{K}^{(n)}), \tag{1.5}$$

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 the variational parameters $\mathcal{K}^{(n)}$ contains those states for which the variational

distribution $q(c; \boldsymbol{x}_n, \mathcal{K}^{(n)}, \tilde{\boldsymbol{\Theta}})$ is non-zero. The optimal values of the variational parameters $\tilde{\boldsymbol{\Theta}}$ are given by the current values of the model parameters $\boldsymbol{\Theta}$ (cf. Lücke, 2019). For $\tilde{\boldsymbol{\Theta}} = \boldsymbol{\Theta}$, the variational free energy becomes

$$\mathcal{F}(\boldsymbol{x}_{1:N}; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}) = \sum_n \log\Big(\sum_{c \in \mathcal{K}^{(n)}} p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})\Big). \tag{1.6}$$

The use of truncated posteriors has enabled the development of numerous efficient algorithms for parameter optimization in probabilistic data models, particularly in the context of large and combinatorially complex latent spaces (e.g, Drefs *et al.*, 2022; Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Mousavi *et al.*, 2023). In this thesis, we investigate algorithms that employ truncated posteriors such as the evolutionary variational optimization (EVO) algorithm (Drefs *et al.*, 2022) in Ch. 2. Furthermore, we develop novel, highly efficient algorithms based on truncated posteriors in Chs. 3 to 5.

1.1.2 Probabilistic Generative Models

In this thesis, we consider and optimize various probabilistic generative models using the conventional EM algorithm or a variational EM version. These models include spike-and-slab sparse coding (SSSC) models, Poisson mixture models (PMMs), Gaussian mixture models (GMMs) with various constraints on the covariance matrices and mixtures of factor analyzers (MFAs), which will be introduced in Ch. 3.

Spike-and-Slab Sparse Coding: Spike-and-slab sparse coding (SSSC; Drefs *et al.*, 2022; Goodfellow *et al.*, 2012; Titsias and Lázaro-Gredilla, 2011) is a probabilistic sparse coding model that combines discrete and continuous latent variables to model sparse representations of data. In the version considered here (Drefs *et al.*, 2022), the continuous latent variables are assumed to follow a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and a full covariance matrix $\boldsymbol{\Psi}$, while the binary latent variables are modeled as independent Bernoulli random variables with activation probabilities π_h for $h \in \{1, \ldots, H\}$, where H denotes the number of discrete latents. The combination of the latents is described by a pointwise multiplication $(\boldsymbol{s} \odot \boldsymbol{z})_h = s_h z_h$ with $\boldsymbol{s} \in \{0, 1\}^H$ and $\boldsymbol{z} \in \mathbb{R}^H$, and the observables $\boldsymbol{x} \in \mathbb{R}^D$ are drawn from a Gaussian distribution. The generative model is defined as:

$$\begin{aligned} p(\boldsymbol{s} \,|\, \boldsymbol{\Theta}) = \prod_{h=1}^{H} \mathcal{B}(s_h; \pi_h), \qquad p(\boldsymbol{z} \,|\, \boldsymbol{\mu}, \boldsymbol{\Psi}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\Psi}), \\ p(\boldsymbol{x} \,|\, \boldsymbol{s}, \boldsymbol{z}, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \sum_h \boldsymbol{W}_h s_h z_h, \sigma^2 \mathbb{1}), \end{aligned} \tag{1.7}$$

where $\boldsymbol{W} \in \mathbb{R}^{D \times H}$ is a weight matrix and σ^2 is the variance of the Gaussian.

Poisson Mixture Models: Poisson mixture models (PMMs) are elementary probabilistic models, in which a non-negative data point $\boldsymbol{x} \in \mathbb{N}^D$ is assumed to be generated by a hidden mixture component $c \in \{1, \ldots, C\}$. The latent variable c follows a categorical distribution and the observed data are drawn from a Poisson distribution. The generative model is defined as:

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) = \prod_{d=1}^{D} \text{Poisson}(x_d; \mu_{cd}), \tag{1.8}$$

where $\pi_c \in [0, 1]$ (with $\sum_c \pi_c = 1$) denote the prior probabilities and $\boldsymbol{\mu}_c \in \mathbb{R}^D_{>0}$ denote the means of the Poisson distributions.

Gaussian Mixture Models: Gaussian mixture models (GMMs) are among the most widely used and well-established tools in probabilistic modeling and unsupervised learning. Similar to PMMs, GMMs assume that each data point is generated by a latent mixture component $c \in \{1, \dots, C\}$, where the latent variable c follows a categorical distribution. However, in contrast to PMMs, GMMs assume that the observed data $\boldsymbol{x} \in \mathbb{R}^D$ are drawn from a Gaussian distribution. The generative model, in its most general form, is defined as:

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c), \tag{1.9}$$

where $\boldsymbol{\mu}_c \in \mathbb{R}^D$ denotes the mean and $\boldsymbol{\Sigma}_c \in \mathbb{R}^{D \times D}$ is the covariance matrix of component c. In this thesis, we consider several variants of GMMs, distinguished by different constraints on the covariance matrices. Commonly used constraints include GMMs with diagonal covariance matrices $\boldsymbol{\Sigma}_c^{\text{diag}} = \text{diag}(\sigma_{c1}^2, ..., \sigma_{cD}^2)$, as well as GMMs with isotropic covariances $\boldsymbol{\Sigma}_c^{\text{iso}} = \sigma_c^2 \mathbb{1}$, where the covariance matrix may additionally be shared across mixture components. These constraints reflect a trade-off between the computational efficiency of GMM optimization and the expressiveness of the model, for example, in terms of its ability to capture correlations in the data (see Ch. 3 for further discussion). In addition, we also consider mixtures of factor analyzers (MFAs), which can be seen as GMMs with low-rank (plus diagonal) covariance matrices (see Sec. 3.2.1 for details).

1.2 Outline

The rest of this thesis is organized as follows: In Ch. 2, we investigate various blind zero-shot denoising algorithms applied to microscopy imaging data from different modalities, particularly at their high-resolution limits. At these limits, microscopy images are affected by various types of noise arising from modality-specific factors in the image acquisition process. Such noise can significantly hinder both expert interpretation and downstream automated analysis. All evaluated algorithms (for details, see Sec. 2.2.2) can be applied directly to a single noisy image, which eliminates the need for extensive training on external datasets and makes them highly data-efficient. Additionally, they operate on image patches, which reduces computational costs compared to full-image processing. The evaluated methods differ both in their underlying noise assumptions, ranging from additive noise models to signal-dependent variances, and in the strategies they employ for efficient training and inference. These strategies include gradient-based optimization, variational approximations, lightweight model architectures, or, in the case of filter-based approaches, no training phase at all. Besides evaluating training time, we compare the algorithms in terms of their denoising performance as well as a metric indicative about image sharpness.

In addition to investigating efficient learning algorithms in the blind zero-shot denoising setting, we build upon prior work (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) and develop, in Ch. 3, a truncated variational optimization method applicable to general GMMs without constraints on the covariance structure. Moreover, the derivation of the variational optimization method does not rely on assumptions specific to GMMs, which lays the foundation for its application to other non-Gaussian mixture models. We integrate this variational optimization method with mixtures of factor analyzers (MFAs), referred to as v-MFA. MFAs can be regarded as a special form of GMMs with low-rank (plus diagonal)

covariance matrices (Ghahramani and Hinton, 1996; McLachlan *et al.*, 2003; Richardson and Weiss, 2018). The MFA approach itself provides an efficient approximation, reducing the complexity of optimizing general GMMs in high-dimensional spaces by modeling data (within each mixture component) along lower-dimensional hyperplanes.

In Ch. 4, we integrate the variational optimization techniques into a distribution-based GSSL framework to mitigate the high computational costs associated with graph-based optimization. The use of truncated posteriors inherently induces sparse graphs. As a result, the proposed variational GSSL approach not only accelerates the distribution learning stage but also improves the efficiency of graph construction and label propagation. We combine this novel GSSL framework with GMMs using diagonal covariance matrices as well as MFAs, denoted as v-GMM$^{\text{d}}$-GL and v-MFA-GL, respectively. The v-MFA algorithm introduced in Ch. 3 is integrated into v-MFA-GL.

In Ch. 5, we adapt the novel v-MFA algorithm for image denoising and evaluate its performance under blind zero-shot conditions. As an initial benchmark, we compare v-MFA with various blind zero-shot denoising algorithms based on DNNs, focusing on both denoising performance and computational efficiency.

In Ch. 6, we provide details on the technical design choices and challenges associated with the CPU-based implementation and parallelization of the v-MFA algorithm introduced in Ch. 3.

Finally, Ch. 7 provides a comprehensive discussion and presents the conclusions of this thesis.

Chapters 2 to 4 consist of contributions in the form of paper manuscripts resulting from collaborative research. These manuscripts have either been published or are under revision in peer-reviewed journals at the time of writing. Chapter 6 was written collaboratively. Full bibliographic references and author contribution statements are provided at the beginning of each respective chapter. A comprehensive list of publications, including those not included in this thesis, is available in the Publication List at the end of the thesis.

Chapter 2

Zero-Shot Denoising of Microscopy Images Recorded at High-Resolution Limits

In this chapter, we investigate a range of zero-shot denoising algorithms applied to microscopy imaging data across different modalities. The evaluated algorithms include elementary and advanced filter-based methods (e.g., Azzari and Foi, 2016b; Dabov *et al.*, 2007; Makitalo and Foi, 2011), DNNs suitable for zero-shot denoising (e.g., Krull *et al.*, 2019a; Lequyer *et al.*, 2022b; Quan *et al.*, 2020a) and probabilistic generative models (e.g., Drefs *et al.*, 2022; Prakash *et al.*, 2021b, and Poisson mixture models, PMMs). These methods differ in their underlying assumptions about the noise, which we discuss in more detail in Sec. 2.2.2 as well as in the strategies they employ for efficient training and inference. The filter-based approaches are the only non-data-driven methods and are inherently efficient, as they do not require a training phase. In contrast, DNNs are typically trained via gradient-based optimization and leverage techniques such as batch-wise learning, which allows for vectorized operations and efficient use of matrix multiplications. These operations are highly amenable to parallelization on GPUs, fostering short runtimes on current hardware. Among the DNN-based methods, Noise2Fast (N2F; Lequyer *et al.*, 2022b) plays a specific role, as it is specifically designed for highly efficient training and employs compact neural network architectures that have demonstrated faster convergence compared to more complex architectures. Similarly, training of PMMs with the EM algorithm also benefits from the relatively simple model structure, which allows for an efficient implementation using vectorized operations and matrix multiplication. The other generative models including Evolutionary Spike-and-Slab Sparse Coding (ES3C; Drefs *et al.*, 2022) and DivNoising (DivN; Prakash *et al.*, 2021b) employ variational optimization techniques to achieve efficient training. ES3C, in particular, uses a variational optimization technique which combines truncated posteriors (see Eq. (1.5)) with evolutionary algorithms.

This chapter has been published as: Sebastian Salwig*, Jakob Drefs* and Jörg Lücke. Zero-shot denoising of microscopy images recorded at high-resolution limits. *PLOS Computational Biology*, 20(6):e1012192, 2024. *joint first authorship.

Below, we present the abstract and author summary of the published paper, followed by a detailed description of the individual contributions made by each author. The remainder of

this chapter corresponds to the sections of the published paper. The associated appendix is provided in Appendix A.

Abstract. Conventional and electron microscopy visualize structures in the micrometer to nanometer range, and such visualizations contribute decisively to our understanding of biological processes. Due to different factors in recording processes, microscopy images are subject to noise. Especially at their respective resolution limits, a high degree of noise can negatively effect both image interpretation by experts and further automated processing. However, the deteriorating effects of strong noise can be alleviated to a large extend by image enhancement algorithms. Because of the inherent high noise, a requirement for such algorithms is their applicability directly to noisy images or, in the extreme case, to just a single noisy image without a priori noise level information (referred to as blind zero-shot setting). This work investigates blind zero-shot algorithms for microscopy image denoising. The strategies for denoising applied by the investigated approaches include: filtering methods, recent feed-forward neural networks which were amended to be trainable on noisy images, and recent probabilistic generative models. As datasets, we consider transmission electron microscopy images including images of SARS-CoV-2 viruses and fluorescence microscopy images. A natural goal of denoising algorithms is to simultaneously reduce noise while preserving the original image features, e.g., the sharpness of structures. However, in practice, a trade-off between both aspects often has to be found. Our performance evaluations, therefore, focus not only on noise removal but set noise removal in relation to a metric which is instructive about sharpness. For all considered approaches, we numerically investigate their performance, report their denoising/sharpness trade-off on different images, and discuss future developments. We observe that, depending on the data, the different algorithms can provide significant advantages or disadvantages in terms of their noise removal vs. sharpness preservation capabilities, which may be very relevant for different virological applications, e.g., virological analysis or image segmentation.

Author Summary. For any image (e.g., recorded using digital photography or electron microscopy), there is a limiting resolution for which the image still looks sharp. Beyond this resolution (by further zooming in), an image becomes increasingly noisy. Resolution limits may be improved technically (bigger lenses, more light, longer exposure times etc.). Alternatively, or additionally, modern denoising algorithms can be applied after the image is recorded. Here we investigate state-of-the-art such algorithms with many of them being developed in the field of Machine Learning. The algorithms work by first learning representations of images (e.g., edges, textures etc.), which they then use to remove the noise from an image. To be applicable at high resolutions, algorithms need to be able to learn from noisy images. If just one noisy image is available for learning without a priori noise level information, the task is also referred to as blind zero-shot denoising. Importantly, we argue that the implicit goal of denoising is noise removal without majorly diminishing image sharpness. All the here investigated blind zero-shot algorithms are consequently evaluated based on both: noise removal and sharpness preservation. We observed salient strengths and weaknesses of different approaches for different image types, which can significantly impact different tasks.

Author Contributions. The idea of benchmarking various state-of-the-art blind zero-shot denoising methods on microscopy imaging data was conceptualized by all authors. All authors contributed to the study design and literature review. Sebastian Salwig (SS)

performed the mathematical derivations related to one of the used denoising methods based on Poisson mixture models (compare Appendix A.1) and implemented that method. Jakob Drefs (JD) in consultation with Jörg Lücke (JL) performed the derivation related to ES3C as reported in the paper and the cited paper (Drefs *et al.*, 2022). SS performed all numerical experiments with contributions from JD, and based on discussion of results with JL. JD and SS prepared the figures and tables with input from JL. All authors wrote the manuscript, discussed the results and their presentation, and approved the final version of the manuscript.

2.1 Introduction

Electron and conventional microscopy can visualize fine details from the micrometer to subnanometer range and are used in biology, medical science, and many other fields. High resolution approaches such as transmission electron microscopy (TEM) have, for instance, significantly contributed to our understanding of how a given virus infects a cell, how it binds to cells and different tissues, and how it spreads in the body (Ivanova *et al.*, 2016; Lamers *et al.*, 2020). For the analysis of virus infections, it is of high importance to appropriately visualize fine details at scales of viruses and cells. Otherwise, recorded structures can be confused with viruses, leading to undesirable misinterpretations of infection scenes, as was discussed, e.g., in the context of SARS-CoV-2 infections (Dittmayer *et al.*, 2020).

The quality of TEM images are influenced by many factors. For instance, only a finite electron dose may be used to avoid damaging the sample. Other factors include stability of the microscope and sample stage as well as electrostatic and magnetic field noise, which may lead to the exclusive use of short exposure times in order to not compromise the resolution (Lee *et al.*, 2014). Therefore, TEM images near their highest resolution contain significant amounts of noise. As reported in Baxter *et al.* (2009), TEM images are affected by three different types of noise: (i) structural noise resulting from the carbon in the background, (ii) Poisson noise due to fluctuations in the number of emitted electrons, and (iii) digitization noise caused by the CCD camera or other scanning devices. Challenges regarding image interpretation and processing due to inherent noise also arise with other microscopy modalities including fluorescence microscopy (FM). In FM images, main noise sources include fluctuations of the photons and digitization noise, which are typically modeled using Poisson and Gaussian distributions, respectively (compare, e.g., Haider *et al.*, 2016; Vonesch *et al.*, 2006).

To address the problem of high noise, algorithms for noise removal are a very established tool (compare, e.g., Wu *et al.*, 2008). These algorithms aim at reducing the noise while preserving the sharpness and structures such as edges, corners or other features of the original image. However, there is often a trade-off between the amount of noise removed on the one hand and the loss of details/sharpness on the other (Crete *et al.*, 2007; Fan *et al.*, 2019). The definition of a 'good' denoised TEM or FM image is very subjective and depends on the application. For instance, strong denoising may achieve a 'good' quantitative assessment in terms of signal-to-noise improvement, but may be unsuitable for analysis from a virologist's perspective because the processed image may be perceived as overly smooth, important details may be lost, or new artifacts may be generated. On the other hand, a strongly denoised and potentially blurred image may be suitable for further image processing tasks such as segmentation, as long as structures of objects important for the task are still preserved (compare Fig. 2.1).

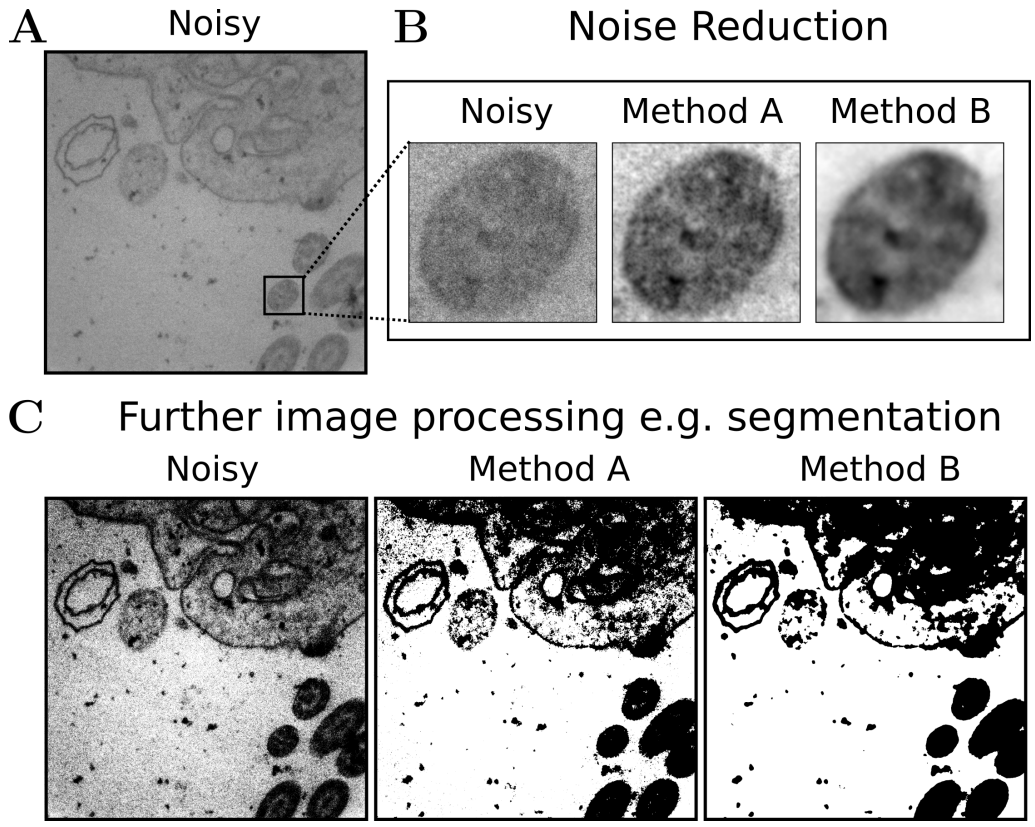

Fig 2.1: Comparison of noise reduction and blur. A: Noisy TEM image of cilia. B: Comparison of an image section of the noisy image, and two denoised versions. While method A reconstructs a sharper image compared to method B (blur effect$_A$ = 0.34, blur effect$_B$ = 0.68, lower is better), method B reduces noise more strongly (PSNR$_A$ = 28.29, PSNR$_B$ = 32.44; higher is better). C: Segmentation masks generated with Otsu's method (Otsu, 1979) based on the noisy image and the reconstructions of method A and B.

Features based on which the large body of existing denoising algorithms can be distinguished include, among others, the strategy applied for noise removal and the amount of a priori information an algorithm requires to denoise a given image. Block-Matching and 3D filtering (BM3D; Dabov *et al.*, 2007), or Weighted Nuclear Norm Minimization (WNNM; Gu *et al.*, 2014), for instance, represent two prominent examples of approaches in the category of non-local image processing methods, that exploit image patch similarities within the image that is supposed to be restored. Both of these algorithms are tailored to Gaussian noise, and they assume the Gaussian noise level to be known a priori. BM3D has been extended in various ways, for instance to be applicable in the case of Poisson noise (Azzari and Foi, 2016b; Makitalo and Foi, 2011). Another filtering approach applicable to microscopy images include the Wavelet Based Background and Noise Subtraction (WBNS) algorithm (Hüpfel *et al.*, 2021). Similar to BM3D or WNMM, this algorithm requires some form of a priori knowledge in order to be applicable (i.e., WBNS requires as input parameter the full width at half maximum of the microscope's point spread function). Data-driven approaches form another category of algorithms, in which denoising models are learned from suitable data, for example large image databases. Widely used are discriminative models that are trained to learn a direct mapping from noisy to non-noisy images using a parameterized function in the form of a deep neural network (DNN). Supervised such algorithms that are

trained based on large sets of paired examples of noisy and respective non-noisy images currently represent the state-of-the-art in a range of standard image restoration benchmarks (e.g., Dong *et al.*, 2019; Tai *et al.*, 2017; Tian *et al.*, 2020; Zhang *et al.*, 2017). Besides attenuation of Gaussian noise, supervised denoising models were found to be highly effective also for a range of more complex types of noise (e.g., Feng *et al.*, 2018; Remez *et al.*, 2017). However, with realistic microscopic image data, as considered in this study, the application of standard supervised algorithms poses a challenge or is not possible due to the lack of sufficiently noise-free training images. Similar applies to unsupervised methods that (at least in practice) leverage clean data for training (e.g., Hurault *et al.*, 2018; Zoran and Weiss, 2011). Waiving the requirement of clean training data for supervised DNNs has been subject to current research: Evaluated on benchmarks with standard test images, Noise2Noise (N2N; Lehtinen *et al.*, 2018), for instance, has been shown to improve the performance of BM3D-based denoisers while being trained on noisy data (and without requiring the noise level a priori). A prerequisite of N2N is, however, that a training dataset with different noisy realizations of the same underlying clean image can be constructed. Such an in practice usually unrealistic assumption motivated follow-up work in the form of Noise2Void (N2V; Krull *et al.*, 2019a) which dropped the requirement of paired noisy images as in N2N training. On standard macroscopic test images, N2V showed competitive denoising performance, yet peak-signal-to-noise ratios (PSNRs) reported were not on par with BM3D and N2N (Krull *et al.*, 2019a). Examples such as N2N or N2V highlight that direct applicability to noisy images has emerged as an increasingly important feature for data-driven denoising algorithms, and ideas exploited by these methods have been taken up in various ways (Bepler *et al.*, 2020; Prakash *et al.*, 2021b; Quan *et al.*, 2020a; Wang *et al.*, 2020), e.g., using a specific form of downsampling to derive a training dataset from a single noisy input etc. (entitled 'chequerboard downsampling'; Noise2Fast; Lequyer *et al.*, 2022b).

A further, very established class of approaches is based on probabilistic generative models which have shown to provide powerful tools for a variety of image enhancement tasks including denoising, deblurring or inpainting (Papyan and Elad, 2016; Parameswaran *et al.*, 2018; Titsias and Lázaro-Gredilla, 2011; Yang and Lee, 2015; Zoran and Weiss, 2011). Generative approaches have also successfully been used, e.g., to decrease EM scan time and electron beam exposure (Ede and Beanland, 2020) using training settings for inpainting or to model a SARS-CoV-2 binder based on 3D coordinates (Eguchi *et al.*, 2022). A foundational element is a mathematical description in terms of a probability density model of the data generating process which incorporates explicit noise assumptions. Furthermore, and unlike discriminate approaches, generative methodologies first learn a probabilistic representation of the data, which can subsequently be used for estimating the underlying non-noisy image (details under Sec. 2.2). Generative approaches are, usually, well suited for task settings with few data or few or no label information, i.e., they can naturally be trained on noisy data in an unsupervised way (Goodfellow *et al.*, 2012; Sheikh *et al.*, 2014; Titsias and Lázaro-Gredilla, 2011; Yang and Lee, 2015; Zhou *et al.*, 2012). Hence, such a property makes them appealing tools in the context of microscopy image denoising.

With this paper, we aim at evaluating the latest developments in the field of unsupervised image denoising using recent microscopy imaging datasets including, most prominently, very recent TEM images of SARS-CoV-2 infection scenes (Laue *et al.*, 2021), TEM images of cilia (Bajić *et al.*, 2018), as well as fluorescence microscopy images of a recent benchmark on unsupervised denoising with deep generative models (Prakash *et al.*, 2021b, 2020) (details under Sec. 2.2.1). The concrete algorithms we here consider are (see Sec. 2.2.2 for details

and Tab. 2.1 for an overview): median filtering (compare, e.g., Wu *et al.*, 2008), BM3D (Dabov *et al.*, 2007) in conjunction with an auxiliary blind noise level estimator (Chen *et al.*, 2015a), BM3D with variance stabilizing transformation (VST+BM3D; Makitalo and Foi, 2011), iterative BM3D with variance stabilizing transformation (I+VST+BM3D; Azzari and Foi, 2016b), N2V, Noise2Fast (N2F; Lequyer *et al.*, 2022b), Self2Self (S2S; Quan *et al.*, 2020a), DivNoising (DivN; Prakash *et al.*, 2021b), Evolutionary Spike-and-Slab Sparse Coding (ES3C; Drefs *et al.*, 2022), and a Poisson Mixture Model (PMM; compare e.g., McLachlan and Peel, 2000). Note that while BM3D itself is a non-blind zero-shot algorithm (as it requires a priori noise level information), all the here considered variants are blind zero-shot approaches.

Our evaluations focus on the setting in which algorithms are provided with only the single noisy microscopy image that has to be denoised, excluding any a priori knowledge about the noise (also referred to as a blind zero-shot setting; compare Chen *et al.*, 2020; Shocher *et al.*, 2018; Soh *et al.*, 2020). Such benchmark conditions have also been considered in the context of other image restoration tasks such as super resolution (Emad *et al.*, 2021; Shocher *et al.*, 2018; Szczotka *et al.*, 2021). Data representations learned based on single observations (such as a single noisy image) often occur naturally whenever data acquisition is difficult or the objects in the image are very rare. As pointed out in work by Laine *et al.* (2021), a data-driven approach is likely to generate artifacts in the reconstructed image if the approach was trained on images with content statistics significantly different from the statistics of the image to be restored. Related to this point, it has (conversely) been argued that learning from single images can be advantageous as intrinsic image statistics contain higher quality information compared to statistics learned from large image corpora (Shocher *et al.*, 2018).

In a variety of denoising benchmarks, performance is commonly assessed based on PSNR values, which are indicative about the mean squared error (MSE) between the processed and target image in relation to the maximal image pixel value. The PSNR is a well established measure and is easy to compute. However, in several cases this evaluation metric performs poorly in reflecting human perception. Wang *et al.* (2004) and Wang and Bovik (2009), for instance, discussed examples in which images with different distortions that obviously present different visual quality, achieved approximately the same MSE (and thus PSNR). Ndajah *et al.* (2010) showed another example where the MSE was not able to properly capture the distortions due to blurring. Motivated by such contributions, this paper presents denoising performance evaluations that do not focus exclusively on noise suppression in terms of MSE/PSNR, but simultaneously inspect measures indicative about both noise suppression and image sharpness. A particular challenge for the here considered datasets with real microscopy images is thereby posed by the absence of ground-truth information (i.e., no 'clean' image data), implying that not only the algorithms themselves but also the evaluation measures need to be applicable on the basis of a single noisy image.

2.2 Materials and Methods

2.2.1 Datasets

We considered three publicly available microscopy image datasets, two recorded with transmission electron microscopy (TEM) and one with fluorescence microscopy (FM). Dataset-specific details will be provided in the following.

Transmission Electron Microscopy Images of SARS-CoV-2 Infected Cell Cultures. Considered were TEM images of ultrathin plastic sections through extracellular SARS-CoV-2 particles in Vero cell cultures (Laue *et al.*, 2021). To compile a dataset for our evaluations, we consulted three publicly available repositories (Laue *et al.*, 2020a,b,c) which provided in total 384 images, from which we selected nine examples (the file names of the selected images are listed in Tab. A.1). Algorithms were separately applied to each of these images. The TEM images had a resolution of 1032×1376 pixels with a pixel size of either 0.64 or 0.54 nm and were stored in 16-bit TIF format; the section thickness was either 65 or 45 nm (see Table 1 of the original publication for details).

Short-Exposure TEM Images of Cilia. The second dataset originated from a TEM image denoising benchmark discussed in Bajić *et al.* (2018) and consisted of a sequence of 100 noisy short exposure TEM images of a scene showing a cilium. We obtained the original data via personal communication with the authors. The images had a resolution of 2048×2048 pixels, and each image depicted a slightly shifted version of the scene. To obtain a less noisy, ground truth-like image, we followed Bajić *et al.* (2018) and first aligned all images of the sequence using rigid registration (Marstal *et al.*, 2016) and subsequently computed the pixel-wise median. Finally, we randomly selected one image from the sequence for our evaluations (in our case the 91st image was selected); the randomly selected noisy image and the pseudo ground-truth image are publicly available (Salwig *et al.*, 2024).

Fluorescence Microscopy Images. As third example, we considered FU-PN2V Convallaria (Krull *et al.*, 2020b; Prakash *et al.*, 2020), FU-PN2V Mouse Actin (Prakash *et al.*, 2020) and FU-PN2V Mouse Nuclei (Prakash *et al.*, 2020) fluorescence microscopy image data, which are publicly available (Krull *et al.*, 2020a; Prakash *et al.*, 2019a,b). In the former two cases, the data consisted of a sequence of 100 images with 1024×1024 pixels, in the latter case of 200 images with 512×512 pixels. To compile a dataset for our evaluations, we followed an earlier suggested procedure and randomly selected one image from each sequence (Prakash *et al.*, 2021b) (in our case, the 75th, 41st, and 177th image were selected for Convallaria, Mouse Actin, and Mouse Nuclei, respectively). A ground truth-like image could be obtained by computing the pixel-wise mean of each image sequence (Prakash *et al.*, 2020). Following publicly available source code associated with this benchmark (Prakash *et al.*, 2021a), we employed full-size versions of each test image (which differs to some extent from the methodology applied in Prakash *et al.* (2020), in which image sections were considered; personal communication with the authors).

2.2.2 Algorithms

We investigated in total ten different blind zero-shot denoising approaches, which differed, among other aspects, in their assumptions about the statistical properties of the noise in the data and their noise removal methodology. Assumptions on noise statistics covered, e.g., additivity or signal-dependency of the noise, explicit expressions for the noise distribution (e.g., Gaussian or Poisson), or non-parametric noise models. The applied noise removal methodologies involved local filtering, non-local filtering, discriminative neural networks and probabilistic generative models. Almost all considered algorithms represented established approaches, for which state-of-the-art results on standard denoising benchmarks had been reported in the respective original publications. For our purposes, we executed the algorithms using publicly available implementations (technical details including hyperparameters are

	Noise is additive	Noise variance is signal-dependent
Filter-based	BM3D	VST+BM3D I+VST+BM3D
Data-driven	N2V DivN S2S N2F ES3C	PMM

Tab 2.1: The noise reduction algorithms considered have different strategies for noise removal and types of noise assumptions. Median filtering is excluded from this table since it has no explicit assumption regarding the noise being additive or signal-dependent. While the DivN approach, in principle, allows for incorporating any suitable imaging noise distribution, here we have followed the publicly available implementation and used a variant of the algorithm in which the noise distribution is described by a GMM. BM3D and ES3C incorporate a Gaussian noise assumption; the VST-based approaches and PMM assume Poisson noise.

described in Appendix A.2). In the following, we provide a description of the different approaches (also see Tab. 2.1 for an overview).

Filtering-based Approaches. As most elementary baseline, we applied median filtering which replaces a given image pixel by the median of the neighboring pixels in a squared window (compare, e.g., Wu *et al.*, 2008). Furthermore, we applied the block-matching and 3D filtering (BM3D; Dabov *et al.*, 2007) algorithm, and its extensions VST+BM3D (Makitalo and Foi, 2011), and I+VST+BM3D (Azzari and Foi, 2016b). These algorithms exploit the assumption that natural images exhibit a high degree of image patch self-similarity: For a given patch of a natural image (i.e., a small image section, also referred to as block), further patches at other locations in the same image can be found which contain similar pixel information (typically in, e.g., l^2 sense). After grouping similar patches together, they are processed via filtering. The original BM3D algorithm assumes Gaussian noise and requires a priori information on the noise level (i.e., on the standard deviation σ). For our purposes, we applied BM3D in conjunction with an auxiliary method for blind noise level estimation, namely the estimator by Chen *et al.* (2015a) which assumes additive white Gaussian noise. VST+BM3D (Makitalo and Foi, 2011) and I+VST+BM3D (Azzari and Foi, 2016b), in contrast, represent algorithms tailored to Poisson noise. They exploit variance-stabilizing transformations (VST) that enable transforming Poisson noise to a noise that can be treated as Gaussian with unit variance (in which case the classical BM3D can readily be applied). Another filtering approach relevant in the context of blind zero-shot denoising of microscopy images is the Median-Gaussian Filtering Framework for Moiré Pattern Noise Removal (Wei *et al.*, 2012) in scanning transmission X-ray microscopy images. The approach was not included in our evaluations, however, as to the best knowledge of the authors, no source code has been made available.

Discriminative Denoising Neural Networks. As a second type of approaches, we applied Noise2Void (N2V; Krull *et al.*, 2019a), Self2Self (S2S; Quan *et al.*, 2020a), and

Noise2Fast (N2F; Lequyer *et al.*, 2022b), all of which employ self-supervised learning paradigms. N2V employs U-Nets (Ronneberger *et al.*, 2015), i.e., convolutional neural networks with a U-shaped architecture, which are trained to learn a mapping between pairs of masked noisy image patches. Specifically, the model learns to map a noisy image patch with a subset of the pixels being discarded to the respective unmodified patch. Since both the input and target training data can be obtained from a single noisy image, N2V is applicable in a blind zero-shot setting. The original publication discusses that the blind-spot strategy is likewise applicable for more conventional training settings with clean targets or with paired noisy images; as pointed out in the paper, such training settings provide advantages for the denoising performance compared to training schemes with only a single noisy image (Krull *et al.*, 2019a). N2V assumes the noise to be additive and conditionally pixel-wise independent given the underlying clean image (the latter of which represents a requirement of the model to avoid learning an identity mapping).

The S2S approach uses DNNs with a bottleneck architecture and partial convolution layers and, similarly to N2V, a form of a blind-spot strategy that enables training on a single noisy image. S2S applies dropout to convolutional layers and image data, which stochastically sets network weights and image pixels to zero, respectively. The masked image and its counterpart, i.e., the image obtained by multiplication with the inverse dropout mask, serve as input and target for the training step. The algorithm computes a restored image by feeding multiple masked versions of a given noisy image through the network and averaging the respective outputs of the model.

N2F uses a convolutional neural network with few hidden layers and as such a comparably simple architecture in relation to N2V and S2S. The training dataset for N2F is derived by downsampling the image to be restored in a way that alternately only preserves even and odd pixels. This process, referred to as 'chequerboard downsampling', is applied in both vertical and horizontal direction, resulting in four images which are split into input and target data; the image to be restored is used as validation data. As discussed in the original publication, the training scheme of N2F was primarily optimized for speed. Lequyer et al. demonstrate that N2F, at the same time, provides improvements compared to approaches such as N2V or Deep Image Prior (DIP; Ulyanov *et al.*, 2018) also in terms of PSNR and SSIM measures (Lequyer *et al.*, 2022b).

Probabilistic Approaches. We considered two shallow and one deep generative model: Evolutionary Spike-and-Slab Sparse Coding (ES3C; Drefs *et al.*, 2022), a standard Poisson Mixture Model (PMM), and DivNoising (DivN; Prakash *et al.*, 2021b). ES3C incorporates a sparse coding data model with a spike-and-slab prior (Goodfellow *et al.*, 2012; Rattray *et al.*, 2009; Titsias and Lázaro-Gredilla, 2011; Yoshida and West, 2010; Zhou *et al.*, 2012) that is trained using evolutionary variational strategies. A foundational objective of the approach is to seek compositional features of the data that, when linearly combined, represent the data sufficiently accurately under an additive white Gaussian noise assumption. Sparse Coding (SC; Olshausen and Field, 1996), more generally, assumes that data can be accurately recovered using a set of few out of many possible compositional features. SC has been studied intensively for image processing and other tasks (e.g., Bornschein *et al.*, 2013; Elad and Aharon, 2006; Mairal *et al.*, 2008; Mousavi *et al.*, 2021; Sheikh *et al.*, 2014).

Unlike SC, mixture models (such as PMMs) assume a single latent per data point for data generation, i.e., data is modeled in terms of a single compositional feature rather than a combination of multiple features. PMMs assume the observable data to be subject to

Poisson distributed noise, and they have been applied in the context of image and audio signal processing applications before (e.g., Irace and Batatia, 2014; Xiang *et al.*, 2021; Zhang and Hirakawa, 2017). The here used PMM-based denoising approach is described in more detail in Appendix A.1.

The DivN approach falls into the category of deep generative models, which can be distinguished from shallow models (such as ES3C or PMM) by their use of DNNs for modeling the data generating process. DivN represents a form of a variational autoencoder (Kingma and Welling, 2014; Rezende *et al.*, 2014) that was originally studied in the context of image denoising applications. The approach incorporates an adjustable model that aims at capturing the noise of the imaging device (i.e., an expression for the distribution of noisy pixels given clean image pixels). The original publication discusses different strategies how such a noise model can be estimated, depending on whether non-noisy or only noisy data is available. One of the former strategies relies on the availability of a sequence of noisy images recorded using the same field of view (i.e., static recording conditions, Krull *et al.*, 2020b). These images, referred to as calibration data, are used to estimate a non-noisy image by averaging the image sequence. As an alternative for a scenario without calibration data, Prakash et al. discuss a strategy referred to as bootstrapping, which leverages auxiliary methodology for estimating a non-noisy image. In both the calibration and the bootstrapping scenario, the noise model is defined in terms of a histogram or a Gaussian mixture model that uses a parameterization depending on both clean and noisy data (Prakash *et al.*, 2020). Furthermore, Prakash et al. discuss a strategy referred to as fully unsupervised which uses a Gaussian noise model with the variance parameterized as a linear function of the modeled clean pixel value (Prakash *et al.*, 2021b). For our purposes, we adapted the examples of the publicly available source code package of DivN and used the bootstrapping method, as the calibration data would violate the blind zero-shot condition. As auxiliary algorithm needed for the bootstrapping method to estimate a non-noisy image, we used N2V. For the learnable noise model of the DivN approach, we chose Gaussian Mixture Models over histograms (also following example code, see Appendix A.2 for further details). Consequently, the variant of DivN that we used in our experiments was tailored to Gaussian noise. Two other related probabilistic approaches include the Stochastically-Connected Random Field algorithm proposed in Haider *et al.* (2016) and the approach developed in Yang and Lee (2015) which is based on a Poisson-Gaussian Contourlet Hidden Markov model. The former of these algorithms combines random graph and field theory; the latter one is designed for low-count fluorescence microscopy images and models the noise as a combination of Poisson and Gaussian probability distributions. Since, to our best knowledge, no source code has been made publicly available for these algorithms, we have not included them in our evaluations.

2.2.3 Evaluation Metrics

For the TEM images of SARS-CoV-2 infected cell cultures, 'clean' reference images were not available, implying that it was not possible to evaluate denoising performance using standard measures such as peak-signal-to-noise ratio (PSNR) or structural similarity (SSIM; Wang *et al.*, 2004). Therefore, in this case, we adopted no-reference metrics that could be evaluated based on a single image (compare e.g., Crete *et al.*, 2007; Sieberth *et al.*, 2016). For the short-exposure TEM images of cilia and the FM image data, on the other hand, 'clean', i.e. ground truth-like, reference images could be estimated (compare Sec. 2.2.1), which enabled the calculation of PSNR values.

Assessment of Noise Suppression. In order to measure noise suppression without ground truth-like reference, we considered the signal-to-noise quantification based on paired hand-labeled signal and background regions as discussed by Bepler *et al.* (2020). Concretely, we followed the procedure described in that publication and labeled M pairs of signal and background regions $\vec{x}_s^{(m)}$ and $\vec{x}_b^{(m)}$ for each test image, respectively, to calculate

$$\mathrm{SNR} = \frac{10}{M} \sum_{m=1}^{M} \log_{10}(s^{(m)}) - \log_{10}(\sigma_b^{2(m)}), \tag{2.1}$$

with $s^{(m)} = (\mu_s^{(m)} - \mu_b^{(m)})^2$, $\mu_s^{(m)}$ and $\mu_b^{(m)}$ denoting the mean of $\vec{x}_s^{(m)}$ and $\vec{x}_b^{(m)}$, respectively, and $\sigma_b^{2(m)}$ denoting the variance of $\vec{x}_b^{(m)}$ (the hand-labeled signal and background regions are depicted in Fig. A.2). In the cases with available ground-truth images, PSNRs were computed as follows:

$$\mathrm{PSNR}(\mathrm{GT}, \mathrm{I}) = 20 \log_{10}(\mathrm{MAX}_{\mathrm{GT}} - \mathrm{MIN}_{\mathrm{GT}}) - 10 \log_{10}(\sum_{\mathrm{d}=1}^{\mathrm{D}} (\mathrm{GT[d]} - \mathrm{I[d]})^2) \tag{2.2}$$

where $\mathrm{MAX}_{\mathrm{GT}}$ and $\mathrm{MIN}_{\mathrm{GT}}$ denote the maximum and minimum pixel amplitudes of the ground-truth image, and GT[d] and I[d] the d-th pixel of the ground-truth and the processed image, respectively.

Assessment of Image Sharpness. For the quantification of image sharpness, we measured the blur effect as suggested by Crete *et al.* (2007). This method analyzes the behavior of neighboring pixels by comparing the original image with a strongly low-pass filtered version of the image (which appears to humans as being significantly blurred). Similarly to the SNR measure by Bepler et al., the blur effect measure can be computed based on a single image; the metric quantifies blur perception in terms of a scalar value ranging from zero to one with lower values corresponding to sharper images (i.e., lower blur perception; see the original publication for details). In the cases with available ground-truth images, we computed the blur effect absolute difference (BAD) between the blur effect value of the ground-truth image and the processed image as follows Kim *et al.* (2022):

$$\mathrm{BAD} = |\text{Blur Effect(GT)} - \text{Blur Effect(I)}| \tag{2.3}$$

To keep information on whether the processed image is sharper or blurrier than the ground-truth image, we have marked points in red or blue if the blur effect value of the ground-truth image is larger or smaller than the blur effect value of the processed image, respectively. Evaluations in terms of SSIM values are provided in Tab. A.3.

2.3 Results

Denoising TEM images of SARS-CoV-2 infected cell cultures. We first investigated blind zero-shot denoising of nine images selected from a publicly available dataset (Laue *et al.*, 2021) containing TEM recordings of SARS-CoV-2 viruses in Vero cell cultures (see Sec. 2.2.1 for details). In order to apply BM3D, VST+BM3D, I+VST+BM3D, N2V, S2S, N2F and DivN to these images, we used the implementations that were made publicly available alongside the official publications, and used the default hyperparameters of those implementations (details in Appendix A.2). All investigated algorithms were applied individually to each test image. Due to the lack of non-noisy reference images, PSNR values

could not be computed. Therefore, we adopted a no-reference metric for noise removal and performed a signal-to-noise quantification based on hand-labeled signal and background regions (Bepler *et al.*, 2020). We contrasted this noise metric with a no-reference blur metric (Crete *et al.*, 2007). The results of the experiment are depicted in Fig. 2.2.

In Fig. 2.2B, a trade-off between SNR and blur effect achieved by the investigated algorithms can be observed. A significant difference exists between the values of S2S and the other investigated methods. While S2S results in the by far highest SNR (indicating the strongest noise suppression), the algorithm also yields the highest blur effect score (indicating most severe blurring of the image). Median filtering achieves the second highest SNR, which is accompanied by the second highest value for the blur effect. After S2S and median filtering, the Poisson-based methods, including PMM, VST+BM3D, and I+VST+BM3D, remove the most noise according to SNR. The SNR and blur effect values of the Poisson-based methods are very close to each other. While ES3C achieves comparable results in terms of the blur effect measure, the algorithm achieves a slightly lower SNR than the Poisson-based methods. The remaining methods, that like ES3C assume additive noise (i.e. DivN, N2F, N2V and BM3D), offer the lowest noise suppression in terms of SNR; at the same time, they achieve the lowest blur effect values.

Denoising of short-exposure TEM images of cilia. As second (more established) benchmark, we considered denoising short exposure TEM images as discussed by Bajić *et al.* (2018). The dataset contains a sequence of noisy short exposure images showing the same scene. As discussed in Bajić *et al.* (2018), a low-noise image can be estimated by registering all images to the first image of the sequence and then computing the pixel-wise median values. For our purposes, we adopted this procedure and considered the pixel-wise median as a pseudo ground-truth image. To consider a blind zero-shot setting, we used a single randomly selected image from the dataset for training and testing the algorithms under investigation (see Sec. 2.2.1 and Appendix A.2 for details related to the data and the initializations and hyperparameters used, respectively). Due to the availability of a pseudo ground-truth image, we could in this case (and unlike in the case of the SARS-CoV-2 TEM data) evaluate PSNR values. To assess image sharpness, we also utilized the pseudo ground-truth image and calculated its blur absolute difference (BAD) w.r.t. the reconstructed image (Kim *et al.*, 2022). To keep information on whether the processed image is sharper or blurrier than the ground-truth image, we have marked points in red or blue to denote whether the blur effect value of the ground-truth image was larger or smaller than the blur effect value of the reconstructed image, respectively. Fig. 2.3 depicts the results of the experiment.

Similarly to the observation made on the SARS-CoV-2 TEM images, S2S achieves the best performance in terms of the noise metric (i.e. PSNR), but the highest value in the blur metric, indicating the strongest blur in the reconstructed image. It is the only algorithm that achieves a worse BAD score compared to the noisy image. The second best methods for noise reduction in terms of PSNR are a group of algorithms including DivN, ES3C, BM3D, N2V, N2F, median filtering and PMM, which also show the best BAD scores. In this group, it is noticeable that, firstly, N2F and N2V provide blur effect values close to the respective value of the ground-truth image and, secondly, that DivN and ES3C achieve the highest PSNR values. The Poisson-based methods VST+BM3D and I+VST+BM3D reduce the least noise and achieve the highest BAD scores alongside S2S and the noisy image. Overall, it is evident that the methods assuming additive noise such as S2S, DivN, ES3C, BM3D,

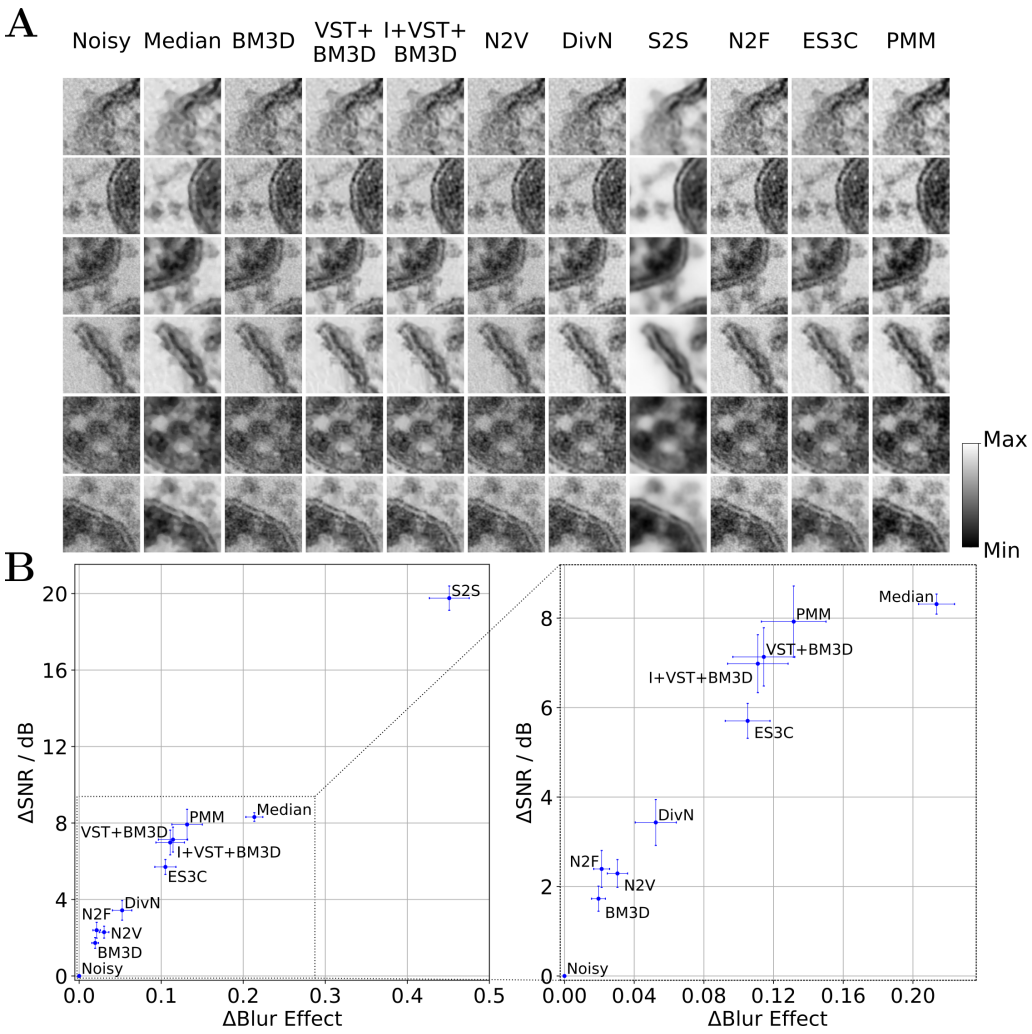

Fig 2.2: Results for blind zero-shot denoising TEM images of SARS-CoV-2 infected cell cultures. A: Qualitative comparison of denoised image sections showing virus and cell structures obtained with the investigated algorithms (image sections are scaled locally to fill the color range). B: Quantification of the denoising performance in terms of SNR in dB based on hand-labeled signal and background regions and blur effect (SNR and blur effect plotted against each other). The left subfigure displays the performance of all algorithms, while the right subfigure provides an enlarged view of the region populated by most algorithms. For both the SNR and blur effect measure, we measured the difference (delta) between the values obtained for the reconstructed and the noisy image and here depict averages together with standard errors of the mean (SEM) over the nine considered test images.

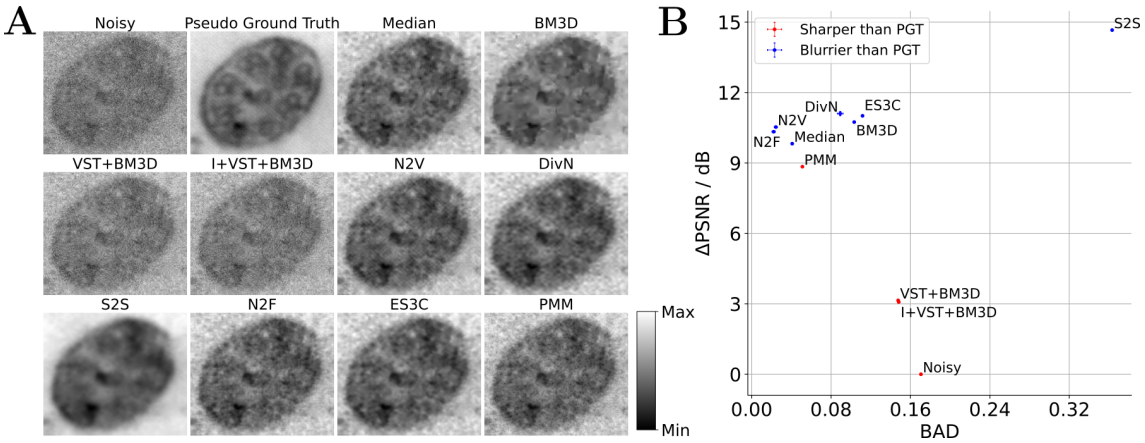

Fig 2.3: Results for blind zero-shot denoising short exposure TEM images of cilia (benchmark adopted from Bajić *et al.*, 2018). A: Qualitative comparison of denoised image sections showing a cilium (image sections chosen to match Fig. 3 in Bajić *et al.* (2018); image sections are scaled locally to fill the color range). B: Quantification of the denoising performance in terms of PSNR in dB and the BAD of the pseudo ground-truth (PGT) image and the denoised reconstruction (PSNR and BAD values plotted against each other). For the PSNR measure, we computed the difference (delta) between the values obtained for the reconstructed and the noisy image and here depict averages together with standard errors of the mean (SEM) over three independent executions of the experiment (note that the SEM values are very small and barely visible in the plot and that they were only evaluated for stochastic algorithms). Points are marked in red or blue if the blur effect value of the PGT image is larger or smaller than the blur effect value of the reconstructed image, respectively. Further details are discussed in the text.

N2F and N2V reduce noise more strongly (considering PSNR values) compared to the Poisson-based methods including PMM, VST+BM3D and I+VST+BM3D. Nevertheless, the PSNR and BAD values of the PMM are comparable to the values of the methods assuming additive noise and significantly better than the values of the other Poisson-based methods VST+BM3D and I+VST+BM3D.

Denoising of fluorescence microscopy images. To consider a different microscopy modality, we adopted the recent fluorescence microscopy image denoising benchmark discussed by Prakash *et al.* (2021b) and applied the algorithms under investigation to the publicly available fluorescence microscopy images FU-PN2V Convallaria, FU-PN2V Mouse Actin, and FU-PN2V Mouse Nuclei. Each of these datasets contains, similarly to the dataset considered in Bajić *et al.* (2018), a sequence of different noisy realizations of a static scene for which a denoised reference can be estimated by taking the pixel-wise mean value of all images in the sequence (Prakash *et al.*, 2020). We adopted the procedure discussed by Prakash *et al.* (2021b), and used a single randomly selected image from each dataset to train and test all algorithms in a blind zero-shot setting (details under Sec. 2.2.1 and Appendix A.2). The results of the experiment are depicted in Fig. 2.4.

For all three images, the best denoising performance in terms of PSNR values is achieved by N2F, which also achieves the best and second best BAD score for Convallaria and Mouse Nuclei, respectively. The second best methods for noise reduction in terms of PSNR are N2V, DivN, ES3C and PMM, with the ranking of these methods being different for each image. Least noise reduction is achieved by VST+BM3D and I+VST+BM3D. Largest performance differences in terms of PSNR across images can be observed for median filtering: For Mouse Actin, the approach achieves similarly high PSNR values compared to other methods (e.g., PMM) while its performance drops significantly for Mouse Nuclei and Convallaria. According to the BAD metric for all images except Mouse Actin, S2S and median filtering result in the most severe blurring. Similarly to the observations made w.r.t. the PSNR measure, the ranking of ES3C, N2F, N2V, DivN, PMM, BM3D, I+VST+BM3D and VST+BM3D in terms of blur effect measure is not consistent across images. Thereby, BM3D, N2F, N2V and also PMM (for Mouse Actin) achieve the best BAD scores in many cases. In additional control experiments, we observed that the performance of DivN in terms of PSNR could significantly be improved by using external calibration images for noise model estimation and a complete image sequence for training (Tab. A.4).

2.4 Discussion

We have investigated a variety of recent algorithms that can be applied to blind zero-shot denoising, i.e. to the task of removing noise from a single high resolution microscopy image without requiring additional a priori knowledge (such as external training data or noise level information). Among the investigated algorithms, approaches such as median filtering, VST+BM3D, I+VST+BM3D, N2V, N2F, S2S, ES3C and PMM can be applied to a single noisy TEM or FM image relatively directly: Median filtering, and the non-local filtering approaches VST+BM3D and I+VST+BM3D are not data-driven and are applicable without prior training. ES3C and PMM are probabilistic models for unsupervised learning and are by definition trainable on noisy data. N2V, S2S and N2F, on the other hand, employ self-supervised training of DNNs and learn discriminative denoising models based on exclusively noisy training images. The further investigated approaches BM3D and DivN were here, unlike the aforementioned algorithms, executed in conjunction with

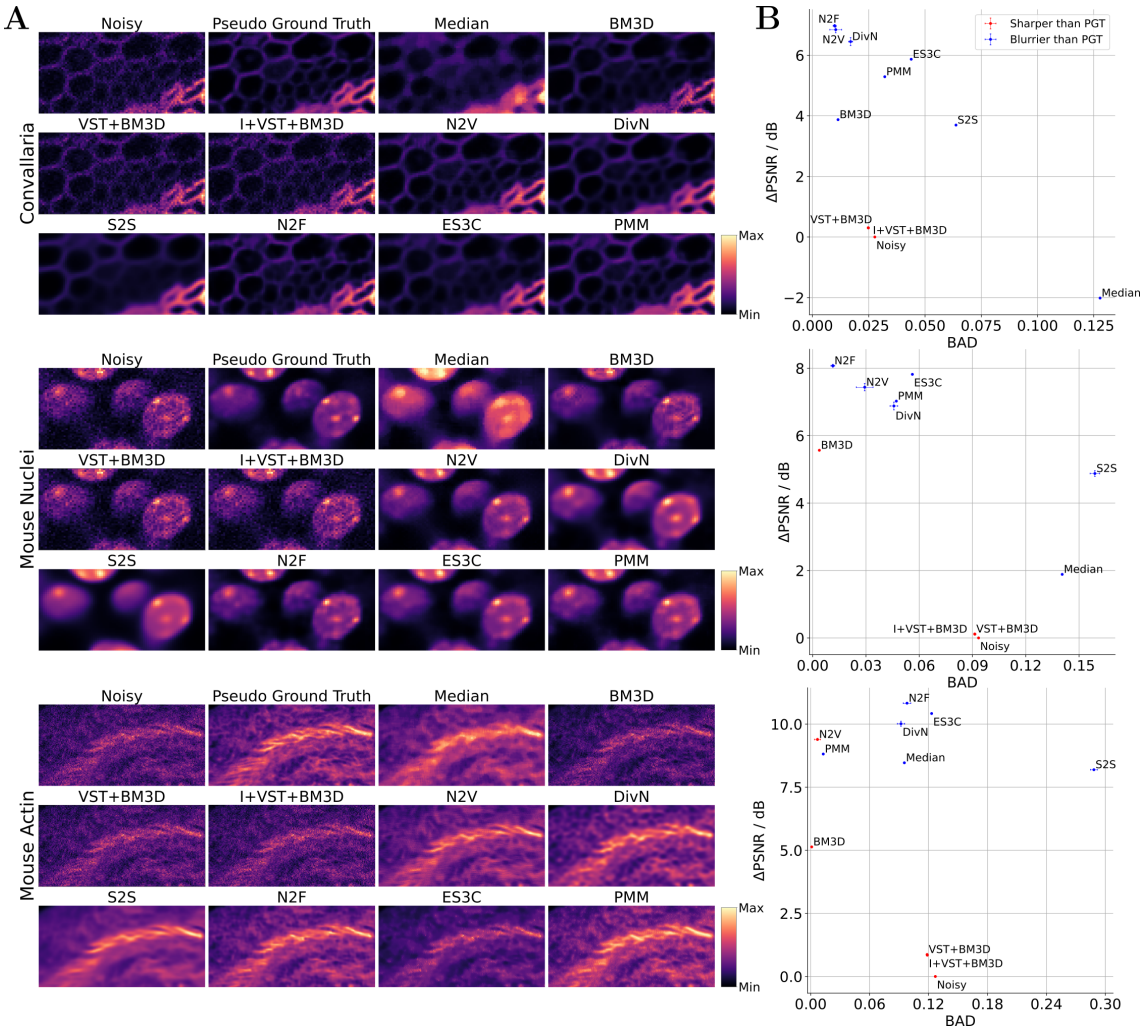

Fig 2.4: Results for blind zero-shot denoising of fluorescence microscopy images (benchmark adopted from Prakash *et al.*, 2021b). A: Qualitative comparison of denoised image sections (image sections and colormap chosen to match Fig 2 in Prakash *et al.* (2021b); image sections are scaled locally to fill the color range). B: Quantification of the denoising performance in terms of PSNR in dB and the BAD of the pseudo ground-truth (PGT) image and the denoised reconstruction (PSNR and BAD values plotted against each other). For the PSNR measure, we computed the difference (delta) between the values obtained for the reconstructed and the noisy image and here depict averages together with standard errors of the mean (SEM) over three independent executions of the experiment (note that SEM were only evaluated for stochastic algorithms). Points are marked in red or blue if the blur effect value of the PGT image is larger or smaller than the blur effect value of the reconstructed image, respectively. Further details are discussed in the text.

auxiliary methodology for noise estimation to be applicable to a single noisy image, following procedures used previously (compare Sec. 2.2.2). The performance of both BM3D and DivN was consequently intertwined with the accuracy of the noise estimation method. That the accuracy of the noise estimation step may turn out to be a crucial factor with potentially decisive impact on the performance for algorithms that rely on such a priori information, has repeatedly been discussed (e.g., Burger *et al.*, 2012; Zhang *et al.*, 2018).

Among the investigated algorithms, those approaches that involve potentially large DNNs with a potentially large number of tunable parameters (e.g., N2V, S2S or DivN), typically increasingly benefit from an increasing amount of training data. N2F plays a particular role in this respect in the sense that it deliberately uses small DNNs which have shown to be advantageous compared to more complex network architectures in blind zero-shot settings in terms of performance and convergence speed (Lequyer *et al.*, 2022b). The investigated (non-deep) generative models (i.e. ES3C or PMM) on the other hand typically rely less on the availability of large amounts of training data to learn appropriate representations (similar applies to the considered median or BM3D-based filtering approaches). Here, the data contained in the noisy image itself was observed to be sufficient. The effectiveness of generative models in image restoration tasks has been demonstrated by numerous contributions (e.g., Drefs *et al.*, 2022, 2023; Sheikh *et al.*, 2014; Titsias and Lázaro-Gredilla, 2011; Ulyanov *et al.*, 2018; Zhou *et al.*, 2012).

For performance evaluations in this study, we focused on two aspects: noise suppression and preservation of image sharpness. To quantify noise suppression, we evaluated signal-to-noise measures, while for image sharpness, we used a no-reference blur metric (cf. Sec. 2.2.3). The results depicted in Figs. 2.2 to 2.4 show that the effectiveness of the investigated algorithms varies across datasets. A potential reason may be given by the match or mismatch between the assumptions made in the algorithms and the statistical properties of the data, which differ, e.g., depending on the microscopy modality. For example, while methods assuming additive noise including N2V, ES3C, BM3D, and DivN (in the here used variant with GMM-based noise model) performed comparably poorly on the SARS-CoV-2 TEM images in terms of the noise suppression metric, they performed significantly better on the short-exposure TEM dataset as well as on the FM data. In contrast, the Poisson-based methods VST+BM3D and I+VST+BM3D showed to be much more effective on the SARS-CoV-2 data compared to the remaining datasets. These observations may be explained by the different dominant noise components in the individual datasets: While the dominant noise in the SARS-CoV-2 data may be assumed to be well-matched by a Poisson distribution, it may, in contrast, be assumed that the dominant noise components in the short-exposure TEM data and the FM data are more closely matched by assuming Gaussian noise (e.g., the digitization noise).

As a further observation, the generative approach PMM performed significantly better on short-exposure TEM and FM data compared to VST+BM3D and I+VST+BM3D, although all three approaches are tailored to Poisson noise. Such an observation may suggest a higher flexibility of the generative approaches in adapting to a noise that may not be exclusively Poisson but a mixture of multiple distributions. In some cases, the performance of PMM was also comparable to N2V, N2F, or DivN, whereby the latter three may be regarded as potentially more suitable approaches for the short exposure TEM and the FM data: N2V, N2F, and DivN assume additive or explicitly Gaussian noise which matches well with the dominant noise source in the short exposure TEM and the FM data.

In control experiments, we observed that algorithm performance could potentially be significantly influenced by varying their hyperparameters. Here, we employed publicly available implementations as far as they were available for the investigated algorithms together with default hyperparameter settings (cf. Appendix A.2). For approaches such as median filtering, ES3C or PMM, which apply a patch-based processing (cf. Fig. A.1), the patch-size showed to be an important hyperparameter (Fig. A.3): For instance, we observed that increasingly large patch sizes resulted in many cases with increases in both the SNR and the blur effect measure, which indicates increasingly strong noise suppression but also an increasing loss of image sharpness. Exceptions to this trend could be observed, e.g., on the FM images Convallaria and Mouse Nuclei, for which PSNR values of ES3C, PMM and median filtering partly decreased when increasing the patch size. In general, increasingly large patches imply an increasingly large amount of context information and an increasingly large number of data estimates per image pixel based on which the reconstructed image is computed (cf. Fig. A.1). At the same time, the complexity and combinatorics of edges, corners, textures and of other structures that can be captured in a patch grows with the patch size. To adequately capture this increase in complexity and combinatorics of structures in the representations of ES3C or PMM, both the number of patches and the number of model parameters have to be increased. However, the number of extractable patches is defined by the image size and the size of the patches. Increasing the number of patches is only possible to a limited extent in a zero-shot setting, e.g., via data augmentation. Therefore, the representations and the reconstructed images of approaches such as ES3C or PMM become increasingly blurred as the patch size increases. The patch size 6×6 employed for the approaches median filtering, ES3C and PMM in Figs. 2.2 to 2.4 may consequently be interpreted as a compromise between both of these aspects (amount of context information and complexity and combinatorics of structures in the patches).

In our experiments, increasingly strong noise reduction was often observed in conjunction with an increasing loss of image sharpness and level of detail. Perhaps most illustrative in this respect may be the results on the SARS-CoV-2 TEM images of the S2S approach that here achieved by far the highest noise reduction and the most severe blurring (Fig. 2.2). Evaluating denoising performance solely based on a metric that is indicative about noise suppression may, based on the here reported results, consequently be argued to be sub-optimal. Strong noise reduction may turn out to be of little use for image analysis by a virologist if details are lost due to strong blur. In such case, it may be more reasonable to resort to an algorithm that reduces noise less strongly but better preserves image sharpness and level of detail. However, if the task is different, e.g., image segmentation, blurring may no longer be an exclusion criterion, and stronger noise reduction coming at the cost of strong blur could be very beneficial as long as no structures required for segment formation are lost (cf. Fig. 2.1). Algorithm runtime represents a further criterion that is potentially relevant when choosing between blind zero-shot approaches. Here, we observed significant runtime differences when executing the implementations of the investigated methods on the considered datasets (Tab. A.2). Particularly strong differences could be observed, for instance, when filtering (median and BM3D-based methods) was compared to methods such as S2S or ES3C: On the one hand, we observed for median filtering or BM3D-based methods runtimes on the order of seconds to a few minutes on a conventional CPU, which was comparably fast. On the other hand, S2S was executed for more than a day on a conventional GPU for the short-exposure TEM data, for instance, and ES3C was executed in parallel on multiple high performance CPUs to achieve runtimes below one hour. Among

the learning-based algorithms, N2F showed to be by far the fastest method, achieving runtimes almost comparable to those of the BM3D-based approaches.

2.4.1 Conclusion

The results of this study suggest that the suitability of a blind zero-shot denoising algorithm for given microscopy imaging data is influenced by several factors, including: (i) the match between algorithm assumptions and statistical properties of the data, (ii) the question to which extent an increase in noise suppression at the cost of a loss of image sharpness may be acceptable, and (iii) requirements regarding computational resources and/or time. The algorithms evaluated here showed individual strengths and weaknesses regarding these aspects. The TEM images reveal that S2S has significantly higher noise suppression but also significantly higher blurring compared to the other methods. In the case of FM images, N2F exhibits the most effective denoising performance in terms of PSNR followed by N2V, ES3C and DivN. N2F, N2V (for the TEM dataset cilia and the FM images) and BM3D (for the FM images) were found to be the methods that most closely matched the blur values of the pseudo ground-truth images. Regarding SARS-CoV-2, it is challenging to make a statement without knowledge of the blur effect values of the ground-truth images. N2F and N2V for the Cilia dataset and the FM images, and the Poisson-based methods including VST+BM3D, I+VST+BM3D and PMM for the SARS-CoV-2 data, appear to offer the best trade-off between noise reduction and sharpness preservation. Our study can provide valuable information about practical characteristics of the investigated methods, which may ideally be instructive for choosing the most suitable algorithm for a particular task, e.g., direct analysis by virologists, or downstream image processing tasks such as segmentation. Furthermore, the study provides typical performance characteristics for different approaches that are developed in a number of different subfields, and developments in all the subfields lead to continuous improvements. The diversity of the different methods studied here may also be of interest from another perspective: Especially methods based on different principles and showing strengths in different aspects and for different tasks can potentially be combined very well by meta-approaches such as boosting. One recent example in this direction is a method that combines denoised images from different methods based on a pixel-wise weight map learned from an unsupervised generative network (Yu *et al.*, 2022), which shows that improved denoising performance can be achieved. Future work may use similar combinations and take denoising as well as sharpness preserving measures into account, or future work may study integrations of, e.g., methods with strong noise reduction and methods with good sharpness preservation. Additionally, it may investigate a combined objective, where optimizing this objective results in strong performance for both criteria.

Chapter 3

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

To address the high computational costs of optimizing Gaussian mixture models (GMMs) with a large number of components in high-dimensional data spaces, we derive in this chapter a highly efficient variational optimization method applicable to general GMMs with arbitrary covariance matrices. Building on prior work (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022), we address their limitations, which arise from derivations relying on assumptions specific to GMMs with isotropic or diagonal covariance matrices. Importantly, our derivation imposes no such restrictive model assumptions, thereby also enabling its application to a broader class of non-Gaussian mixture models (compare Secs. 3.2.2 and 3.2.3). We integrate the proposed variational optimization method with mixtures of factor analyzers (MFAs), referred to as v-MFA. The MFA approach itself provides an efficient approximation by modeling each mixture component along lower-dimensional hyperplanes, which reduces the complexity of optimizing general GMMs in high-dimensional spaces (compare Appendix B.1.1).

This chapter is currently under review for publication as: Sebastian Salwig*, Till Kahlke*, Florian Hirschberger, Dennis Forster and Jörg Lücke. Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters. *joint first authorship.

A preprint is published as: Sebastian Salwig*, Till Kahlke*, Florian Hirschberger, Dennis Forster and Jörg Lücke. Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters. arXiv preprint arXiv:2501.12299, 2025. *joint first authorship.

Below, we present the abstract of the paper, followed by a detailed description of the individual contributions made by each author. The remainder of this chapter corresponds to the sections of the paper. The associated appendix is provided in Appendix B.

Abstract. Gaussian Mixture Models (GMMs) range among the most frequently used models in machine learning. However, training large, general GMMs becomes computationally prohibitive for datasets 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. Secondly, 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.

Author Contributions. The idea of generalizing the variational EM algorithm introduced in Hirschberger *et al.* (2022) for application to general mixture models and combining it with mixtures of factor analyzers (MFAs) was initially conceptualized by Florian Hirschberger (FH), Dennis Forster (DF) and Jörg Lücke (JL). Preliminary numerical studies were conducted by FH, with input from JL and DF, producing initial results that were not included in the current version of the manuscript. The project was subsequently taken over by Till Kahlke (TK) and Sebastian Salwig (SS).

The literature review was carried out by all authors. Initial theoretical considerations to generalize the updates of the g_c sets were done by DF with contributions by JL. Using aspects of the preliminary investigations, TK and SS worked out the estimate of the KL-divergence used in the paper by using their own empirical investigations and their own theoretical investigations (the latter done jointly with JL). TK and SS performed the numerical experiments with input from JL (partly inspired by the initial experiments from FH). Figures and tables were created by TK, SS and DF with input from JL. The manuscript was written by TK, SS and JL with contributions from FH and DF. All authors discussed the results and approved the final version of the paper.

The software was developed collaboratively. An initial implementation of the variational EM algorithm was designed and written by FH. The algorithm was primarily implemented in C++ using the Blaze and Boost libraries and included a Python wrapper using pybind11. This initial version focused on efficient serial execution (on a single CPU thread). Building on this version, TK and SS subsequently developed a new version of the implementation, transitioning from Blaze to the Eigen library and placing particular emphasis on parallelizing the variational EM algorithm (within a CPU node). Further contributions of SS and TK include improvements in overall performance, usability, accessibility and the Python interface.

3.1 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 dataset 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., Bouguila and Fan, 2020; Robin and Scrucca, 2023; Tian *et al.*, 2019; Zoran and Weiss, 2011).

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., Escobar and West, 1995; Li and Barron, 1999; Maz'ya and Schmidt, 1996; Parzen, 1962; Zeevi and Meir, 1997).

In the density modeling context, flexible component parametrization and the number of components are crucial for the approximation quality (e.g., Li and Barron, 1999; Zeevi and Meir, 1997). However, if GMMs are applied to the large-scale datasets 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 (Exarchakis *et al.*, 2022; Hirschberger *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 (Bei and Gray, 1985; Cheng *et al.*, 1984; Elkan, 2003), random projections (Chan and Leung, 2017), or dimensionality reduction approaches (Bouveyron *et al.*, 2007; Hertrich *et al.*, 2022; Liu *et al.*, 2022; Richardson and Weiss, 2018).

Still another line of research to reduce the optimization cost of GMMs aims at reducing the number of data points. Such reduced datasets are known as coresets (e.g., Feldman *et al.*, 2011; Har-Peled and Mazumdar, 2004; Lucic *et al.*, 2018) 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 (Exarchakis *et al.*, 2022; Hirschberger *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 dataset (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. Drefs *et al.*, 2022; Lücke and Eggert, 2010) 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 datasets.

Additionally, we will make use of advanced seeding approaches (e.g., Arthur and Vassilvitskii, 2007; Bachem *et al.*, 2016a,b; 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, in general, used to reduce optimization complexity (Jordan *et al.*, 1999; Neal and Hinton, 1998), and has repeatedly been applied to mixture models (e.g., Forster *et al.*, 2018; Neal and Hinton, 1998; Shelton *et al.*, 2017). In the case of GMMs with diagonal covariances, it has recently been shown (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) 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. 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 components (Exarchakis *et al.*, 2022; Hirschberger *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 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. For $H = D$ general GMMs are recovered[1]. For many types of data, H can be much smaller than D, however, and the complexity of an EM step is reduced from $\mathcal{O}(NCD^2)$ to $\mathcal{O}(NCDH)$. Due to their reduced complexity in D, MFAs are applicable to high-dimensional data (e.g., Kock *et al.*, 2022), and they can on such data 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.

The training principles and the derivation of v-MFA are described in Sec. 3.2, and its numerical evaluation is described in Sec. 3.3.

[1] although the GMM is overparameterized in this case

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 Secs. 3.2.1 to 3.2.3. Finally, the implementation of the complete, variational EM algorithm is described in Sec. 3.2.4.

3.2.1 Variational Optimization of MFAs

In the Mixture of Factor Analyzers (MFA) model (Ghahramani and Hinton, 1996; Richardson and Weiss, 2018), a data point $\boldsymbol{x} \in \mathbb{R}^D$ is modeled by a hidden mixture component $c \in \{1, \ldots, C\}$ and a hidden factor $\boldsymbol{z} \in \mathbb{R}^H$. 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 (McLachlan and Peel, 2000; McLachlan *et al.*, 2003) with a factor loading matrix $\boldsymbol{\Lambda}_c \in \mathbb{R}^{D \times H}$. The generative model is given by

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{z} \,|\, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}), \qquad p(\boldsymbol{x} \,|\, \boldsymbol{z}, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c, \boldsymbol{D}_c), \quad (3.1)$$

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

An alternative perspective on the MFA model involves considering it as a GMM with low-rank covariance matrix $\boldsymbol{\Sigma}_c = \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^\top + \boldsymbol{D}_c$. In this case, the generative model is expressed as

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^\top + \boldsymbol{D}_c). \quad (3.2)$$

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

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

$$\mathcal{L}(\boldsymbol{x}_{1:N}; \boldsymbol{\Theta}) = \log \big(p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\Theta})\big) = \sum_{n=1}^{N} \log \Big(\sum_{c=1}^{C} \pi_c \, \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^\top + \boldsymbol{D}_c) \Big). \quad (3.3)$$

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; Jordan *et al.*, 1999; Neal and Hinton, 1998). 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 $\boldsymbol{\Theta}$ in the M-step. In this study, we employ a variational version of the EM algorithm, which uses truncated posterior distributions (e.g., Drefs *et al.*, 2022; Lücke and Eggert, 2010; Shelton *et al.*, 2017) as its family of variational distributions. Concretely, we use variational distributions $q_n(c; \tilde{\boldsymbol{\Theta}})$ defined by

$$\forall n = 1, ..., N: \quad q_n(c; \tilde{\boldsymbol{\Theta}}) := q(c; \boldsymbol{x}_n, \mathcal{K}^{(n)}, \tilde{\boldsymbol{\Theta}}) = \frac{p(c, \boldsymbol{x}_n \,|\, \tilde{\boldsymbol{\Theta}})}{\sum_{\tilde{c} \in \mathcal{K}^{(n)}} p(\tilde{c}, \boldsymbol{x}_n \,|\, \tilde{\boldsymbol{\Theta}})} \, \delta(c \in \mathcal{K}^{(n)}), \quad (3.4)$$

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 $q_n(c; \tilde{\boldsymbol{\Theta}})$ 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 (Eq. (3.4)), we obtain as free energy objective

$$\mathcal{F}(\boldsymbol{x}_{1:N}; \boldsymbol{\mathcal{K}}, \tilde{\boldsymbol{\Theta}}, \boldsymbol{\Theta}) := \sum_{n=1}^{N} \left(\sum_{c=1}^{C} q_n(c;\, \tilde{\boldsymbol{\Theta}}) \log \left(\pi_c \, \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^\top + \boldsymbol{D}_c) \right) + \mathcal{H}\left[q_n\right] \right), \quad (3.5)$$

where $\boldsymbol{\mathcal{K}}$ is the collection of all index sets $\mathcal{K}^{(n)}$ and $\mathcal{H}\left[q_n\right] = -\mathbb{E}_{q_n}[\log q_n(c;\, \tilde{\boldsymbol{\Theta}})]$ is the Shannon entropy. The 'hard' zeros used by the truncated distributions mean that all sums over components C effectively only have to evaluate C' non-zero summands. 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 $\tilde{\boldsymbol{\Theta}}$ are the current values of the model parameters $\boldsymbol{\Theta}$, which can be shown in general (Lücke, 2019) and which, therefore, also applies for the MFA model. For $\tilde{\boldsymbol{\Theta}} = \boldsymbol{\Theta}$ the free energy simplifies to (compare, e.g., Drefs *et al.*, 2022; Lücke, 2019):

$$\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}) := \mathcal{F}(\boldsymbol{x}_{1:N}; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}, \boldsymbol{\Theta}) = \sum_{n=1}^{N} \log\Big(\sum_{c \in \mathcal{K}^{(n)}} p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) \Big). \quad (3.6)$$

As could be deduced from Eq. (3.6), optimizing the MFA model using variational EM requires to repeatedly evaluate joint probabilities $p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$ or, equivalently, log-joints. In previous work (Exarchakis *et al.*, 2022; Hirschberger *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 B.1.1 and B.1.2, respectively.

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 (Exarchakis *et al.*, 2022; Hirschberger *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 $p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$ defined by the MFA generative model in Eq. (3.2), and the free energy $\mathcal{F}$ for index sets $\boldsymbol{\mathcal{K}}$. 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 $p(\tilde{c}, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) > p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$.

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 B.1.3.

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

$$\mathcal{K}^{(n)}_{\text{opt}} = \{c \mid p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta}) \text{ is among the } C' \text{ largest joints}\}, \tag{3.7}$$

i.e., the optimal sets $\mathcal{K}^{(n)}_{\text{opt}}$ contain those C' components with the highest joint probabilities (also compare Forster *et al.*, 2018; Shelton *et al.*, 2017). 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 Eq. (3.7) 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[2], that includes candidate components $\tilde{c}$ which may replace a $c \in \mathcal{K}^{(n)}$ to increase the free energy for each data point $\boldsymbol{x}_n$. 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 $\boldsymbol{x}_n$. Given $\mathcal{S}^{(n)}$ the updated $\mathcal{K}^{(n)}$ is defined by:

$$\mathcal{K}^{(n)} = \{c \mid p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta}) \text{ is among the } C' \text{ largest joints for all } c \in \mathcal{S}^{(n)}\}. \tag{3.8}$$

The index set $\mathcal{K}^{(n)}$ of Eq. (3.8) can consequently be determined by evaluating all joints with $\boldsymbol{x}_n$ 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 Eq. (3.8) never decreases the free energy $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$. However, this condition on the search spaces $\mathcal{S}^{(n)}$ is not sufficient to warrant an efficient increase of the free energy. For a high efficiency, components $\tilde{c} \in \mathcal{S}^{(n)}$ that are not in $\mathcal{K}^{(n)}$ have to be likely to result in larger joints, i.e., it has to be sufficiently likely for $\tilde{c} \in \mathcal{S}^{(n)}$ that $p(\tilde{c}, \boldsymbol{x}_n \mid \boldsymbol{\Theta}) > p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta})$ with $\tilde{c} \notin \mathcal{K}^{(n)}$ and $c \in \mathcal{K}^{(n)}$.

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 $\{1, \ldots, C\}$. However, as C increases, the probability of finding a new component $\tilde{c}$ with high joint $p(\tilde{c}, \boldsymbol{x}_n \mid \boldsymbol{\Theta})$ 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.

3.2.3 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

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

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. Using the sets g_c, a search space $\mathcal{S}^{(n)}$ for a given data point is defined as the following union:

$$\mathcal{S}^{(n)} := \bigcup_{c \in \mathcal{K}^{(n)}} g_c \,. \tag{3.9}$$

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 reached in the case when all g_c are disjunct (in that case its size is given by $S = |\mathcal{K}^{(n)}|\,|g_c| = C'G$)[3]. In general the index sets g_c may intersect, however, leading to smaller $\mathcal{S}^{(n)}$.

The construction of the search space by Eq. (3.9) 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 $\boldsymbol{x}_n$. 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 3.1 offers an illustration of the sets g_c and the construction of $\mathcal{S}^{(n)}$. Figure 3.1**A** shows a schematic representation, Figure 3.1**B** visualize $\mathcal{K}^{(n)}$ and g_c, and Figure 3.1**C** illustrates $\mathcal{S}^{(n)}$ in data space. Figure 3.1 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 (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) unsuitable for finding relevant components for a given component. Figure 3.2 highlights that for general covariance matrices, small 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:

$$D_{\mathrm{KL}}\left[p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) \;||\; p(\boldsymbol{x} \,|\, \tilde{c}, \boldsymbol{\Theta})\right] = \mathbb{E}_{p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta})}\left[\log \frac{p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta})}{p(\boldsymbol{x} \,|\, \tilde{c}, \boldsymbol{\Theta})}\right]. \tag{3.10}$$

Although there exists an analytically tractable form of the KL-divergence for Gaussian distributions, evaluating Eq. (3.10) introduces additional computational overhead, and the divergence has to be computed for all $(c, \tilde{c})$ pairs. Thus, we aim to efficiently estimate

[3] In the final v-MFA algorithm, we additionally add one component uniformly at random to the search spaces, resulting in $S = C'G + 1$ (compare Alg. 1).

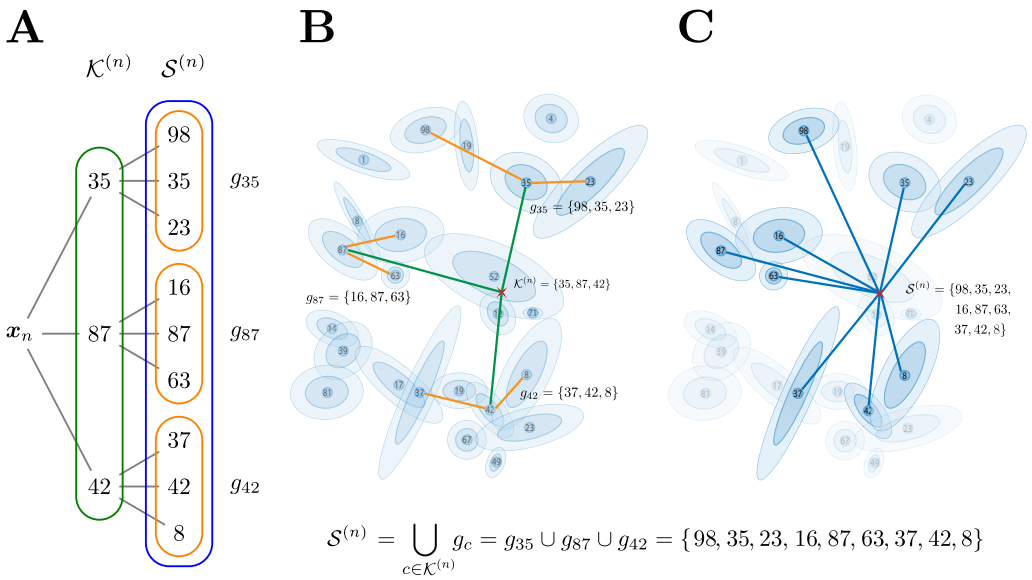

Fig 3.1: Construction of $\mathcal{S}^{(n)}$ for a data point $\boldsymbol{x}_n$. Subfigure **A** shows a schematic representation while **B** visualizes $\mathcal{K}^{(n)}$ and g_c, and **C** visualizes $\mathcal{S}^{(n)}$ in data space. In this example, the set $\mathcal{K}^{(n)} = \{35, 87, 42\}$ contains $C' = 3$ component indices for $\boldsymbol{x}_n$, 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 forms the search space $\mathcal{S}^{(n)} = \{98, 35, 23, 16, 87, 63, 37, 42, 8\}$, as indicated by the blue box in **A** and blue lines in **C**. None of the sets are optimal for the data point $\boldsymbol{x}_n$ yet. The optimization procedure will be explained in the following.

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. To achieve this, we employ a finite sample approximation of Eq. (3.10). Instead of drawing samples from $p(\boldsymbol{x} \mid c, \boldsymbol{\Theta})$, we estimate the KL-divergence by using the data points currently assigned to component c. This gives rise to the introduction of a partitioning of the dataset into the following sets:

$$\mathcal{I}_c = \{n \mid c = c_n\} \quad \text{with} \quad c_n = \underset{c' \in \mathcal{K}^{(n)}}{\operatorname{argmax}}\, p(c' \mid \boldsymbol{x}_n, \boldsymbol{\Theta}) = \underset{c' \in \mathcal{K}^{(n)}}{\operatorname{argmax}}\, p(c', \boldsymbol{x}_n \mid \boldsymbol{\Theta}). \tag{3.11}$$

Given the sets $\mathcal{I}_c$, the KL-divergence between the probability distributions of a component c and $\tilde{c}$ can be estimated as follows (see Appendix B.1.4 for details):

$$D_{\mathrm{KL}}\left[p(\boldsymbol{x} \mid c, \boldsymbol{\Theta}) \;||\; p(\boldsymbol{x} \mid \tilde{c}, \boldsymbol{\Theta})\right] \approx \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log \frac{p(\boldsymbol{x}_n \mid c, \boldsymbol{\Theta})}{p(\boldsymbol{x}_n \mid \tilde{c}, \boldsymbol{\Theta})} \delta(\tilde{c} \in \mathcal{S}^{(n)}) =: D_{c\tilde{c}} \tag{3.12}$$

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

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. Due to $\delta(\tilde{c} \in \mathcal{S}^{(n)})$ in Eq. (3.12), 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

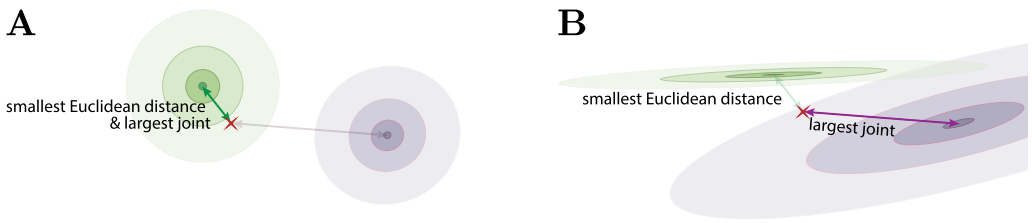

Fig 3.2: Comparison of Euclidean distances and joints. In **A** and **B**, the component centers are identical and the same data point is marked with a red cross. For isotropic, equally weighted components (**A**), the smallest Euclidean distance correspond to the largest joint between components and the data point. However, for general covariance matrices (**B**), this is no longer the case.

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 (3.12) 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.,

$$g_c = \{\tilde{c} \mid D_{c\tilde{c}} \text{ is among the } G \text{ lowest values}\}. \tag{3.13}$$

The definition of $D_{c\tilde{c}}$ in Eq. (3.12) ensures that no additional computations of $p(\boldsymbol{x}_n \mid \tilde{c}, \boldsymbol{\Theta})$ are required beyond those that anyway have to be performed in a partial E-step (see Sec. 3.2.4 for details). As a result, $D_{c\tilde{c}}$, and thus the sets g_c, are consequently very efficiently computable. A potential drawback 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 B.1.4). 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 B.1.4.1 and B.2.2.1. While the here defined method (based on Eq. (3.12)) is especially advantageous in the general case with intra-component correlations, we also observed improvements for the uncorrelated case.

3.2.4 Algorithmic Realization of v-MFA

While Secs. 3.2.1 to 3.2.3 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 B.1.2.

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 3.3 visualizes the iterative procedure for $\mathcal{K}^{(n)}$ optimization.

(2) The partition $\mathcal{I}_{1:C}$ of the dataset is computed.

(3) The partition and the previously computed joints are used to determine the sets $g_{1:C}$, utilizing
$\log p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta}) = \log p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) - \log p(c \,|\, \boldsymbol{\Theta})$. $D_{c\tilde{c}} = \tilde{D}_{c\tilde{c}}/N_{c\tilde{c}}$ 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 $q_n(c; \boldsymbol{\Theta})$ are computed by normalizing of $p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$.

Algorithm 1: Variational E-step

for $n = 1 : N$ **do**

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

for $n = 1 : N$ **do**

2
- $c_n = \underset{c \in \mathcal{K}^{(n)}}{\operatorname{argmax}}\, p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$;
- $\mathcal{I}_{c_n} = \mathcal{I}_{c_n} \cup \{n\}$;

for $c = 1 : C$ **do**

3
- **for** $n \in \mathcal{I}_c$ **do**
  - **for** $\tilde{c} \in \mathcal{S}^{(n)} \setminus \{c\}$ **do**
    - $\tilde{D}_{c\tilde{c}} = \tilde{D}_{c\tilde{c}} + \log p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta}) - \log p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})$;
    - $N_{c\tilde{c}} = N_{c\tilde{c}} + 1$;
- $g_c = \{\tilde{c} \,|\, (\tilde{D}_{c\tilde{c}}/N_{c\tilde{c}})$ is among the $G - 1$ 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**
  - $q_n(c; \boldsymbol{\Theta}) = p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) / \sum_{\tilde{c} \in \mathcal{K}^{(n)}} p(\tilde{c}, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$;

The computational complexity of a single variational E-step is $\mathcal{O}(NSDH)$, dominated by the joint computations in block 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 B.1.5.

Initialization of model parameters: To preserve the algorithmic complexity of Alg. 1, initialization methods should exhibit complexities that are comparable 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-MC^2 (Bachem *et al.*, 2016a) as an efficient initialization method for initial

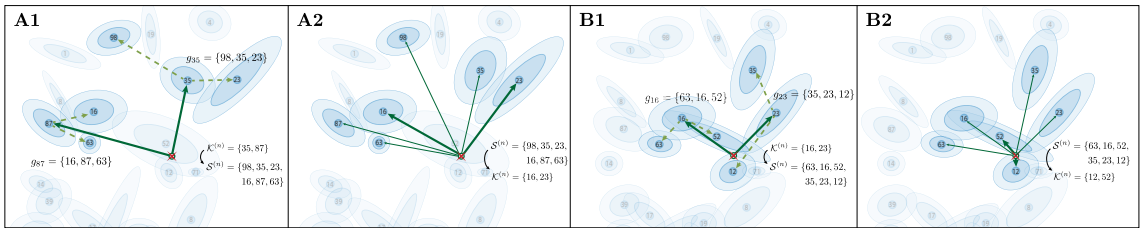

Fig 3.3: Update of $\mathcal{K}^{(n)}$ over two iterations. In **A1**, a red cross marks a data point $\boldsymbol{x}_n$ with an initial suboptimal $\mathcal{K}^{(n)} = \{35, 87\}$ (we are here for simplicity only using $C' = 2$) and $G = 3$ replacement candidates for each of those components: $g_{35} = \{98, 35, 23\}$ and $g_{87} = \{16, 87, 63\}$. Also these replacement candidates are still suboptimal. Subsequently, in **A2**, the joints $p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$ to all components in the search space $\mathcal{S}^{(n)} = g_{35} \cup g_{87} = \{98, 35, 23, 16, 87, 63\}$ 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 12 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.

values for the means $\boldsymbol{\mu}_c$. To initialize the diagonal variances $\boldsymbol{D}_{1:C}$, we compute the variance along each dimension across all data points as follows:

$$\boldsymbol{D}_c = \boldsymbol{D}_{\text{init}} \quad \forall c \qquad \text{with } \boldsymbol{D}_{\text{init}} = \text{diag}(\sigma_1^2, \cdots, \sigma_D^2) \text{ and } \sigma_d^2 = \text{var}(x_{1:N,d}), \tag{3.14}$$

where $x_{n,d}$ is the d-th entry of $\boldsymbol{x}_n$. Regarding the factor loading matrices $\boldsymbol{\Lambda}_c$, we used 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-MC^2, 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 $\{1, \ldots, C\}$ (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 Alg. 2. Following initialization, the algorithm performs variational E-steps in a *warm-up* phase until the free energy $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ has converged. Subsequently, the main training process alternates between variational E-steps and M-steps until $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ has converged again. As convergence criterion for both loops, we consider $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ to have converged after iteration t if

$$\frac{\mathcal{F}^{(t)} - \mathcal{F}^{(t-1)}}{\mathcal{F}^{(t-1)}} < \varepsilon, \tag{3.15}$$

where $\mathcal{F}^{(t)}$ is the free energy $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ of Eq. (3.6) at iteration t (cf. Hirschberger *et al.*, 2022). In all numerical experiments we choose the convergence threshold $\varepsilon = 10^{-4}$ unless specifically stated otherwise.

Algorithm 2: v-MFA

initialize $\pi_{1:C}$, $\boldsymbol{\mu}_{1:C}$, $\boldsymbol{\Lambda}_{1:C}$, $\boldsymbol{D}_{1:C}$, $\mathcal{K}_{1:N}$ and $g_{1:C}$ (see text for details);
repeat
1 | *variational E-step:* update $\mathcal{K}_{1:N}$, $g_{1:C}$ using Alg. 1;
until $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ *has converged*;

---

repeat
2 | *variational E-step:* update $\mathcal{K}_{1:N}$, $g_{1:C}$ using Alg. 1;
| *variational M-step:* update $\pi_{1:C}$, $\boldsymbol{\mu}_{1:C}$, $\boldsymbol{\Lambda}_{1:C}$ and $\boldsymbol{D}_{1:C}$ (see Appendix B.1.2 for details);
until $\mathcal{F}(\boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ *has converged*;

3.3 Results

To assess the efficiency and effectiveness of the v-MFA algorithm, we conducted numerical experiments organized into three sections. In Sec. 3.3.1, 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 Sec. 3.3.2 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, denoted as k-means+*FA*. Finally, in Sec. 3.3.3, we extend the analysis to dataset 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. Further details on the numerical experiments are provided in Appendix B.2.1, and further control experiments are reported in Appendix B.2.2.

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.2.1.3). The datasets have frequently been applied in different contexts, and they were used previously in settings similar to the one of interest here (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Richardson and Weiss, 2018). 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 datasets 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. Exarchakis *et al.*, 2022; Hirschberger *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 runtimes hold significant

practical relevance, albeit this measure is inherently contingent upon implementation details and on the specification of used hardware.

For the models under consideration, we assess the algorithms' effectiveness using the negative log-likelihood (NLL), which represents a canonical quality measure for probabilistic models. In experimental settings where the em-MFA algorithm can be used as a reference, we employ the relative NLL given by

$$\text{rel. NLL} = \frac{\text{NLL}_{\text{algo}} - \text{NLL}_{\text{em-MFA}}}{\text{NLL}_{\text{em-MFA}}}, \tag{3.16}$$

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.

3.3.1 Scalability

The v-MFA algorithm exhibits a reduced complexity per iteration compared to em-MFA (cf. Sec. 3.2.1). 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 dataset size while increasing C could simplify optimization, typically resulting in a reduced number of iterations. To mitigate this effect, we simultaneously increase the datasets size alongside C. We constructed these datasets 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 dataset was used. Our scalability evaluation differs in detail from previous evaluations (e.g., Hirschberger *et al.*, 2022), but we also adjusted dataset 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 datasets, 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 Sec. 3.2, $S = C'G + 1$ 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 datasets and component numbers. For the effect of different hyperparameters on the results, we refer to the experiments in Sec. 3.3.2. The v-MFA and em-MFA algorithms iterate (variational) EM updates until a convergence criterion is reached (see Eq. (3.15)).

The results of the scalability analysis are summarized in Fig. 3.4. To quantify the scalability with C, we fitted regressions of the form $f(C) = b\,C^a$. This corresponds to linear functions $\log(f(C)) = \tilde{b} + a \log(C)$ in log-log plots. Hence, the slopes in Fig. 3.4 provide the scaling exponents for C.

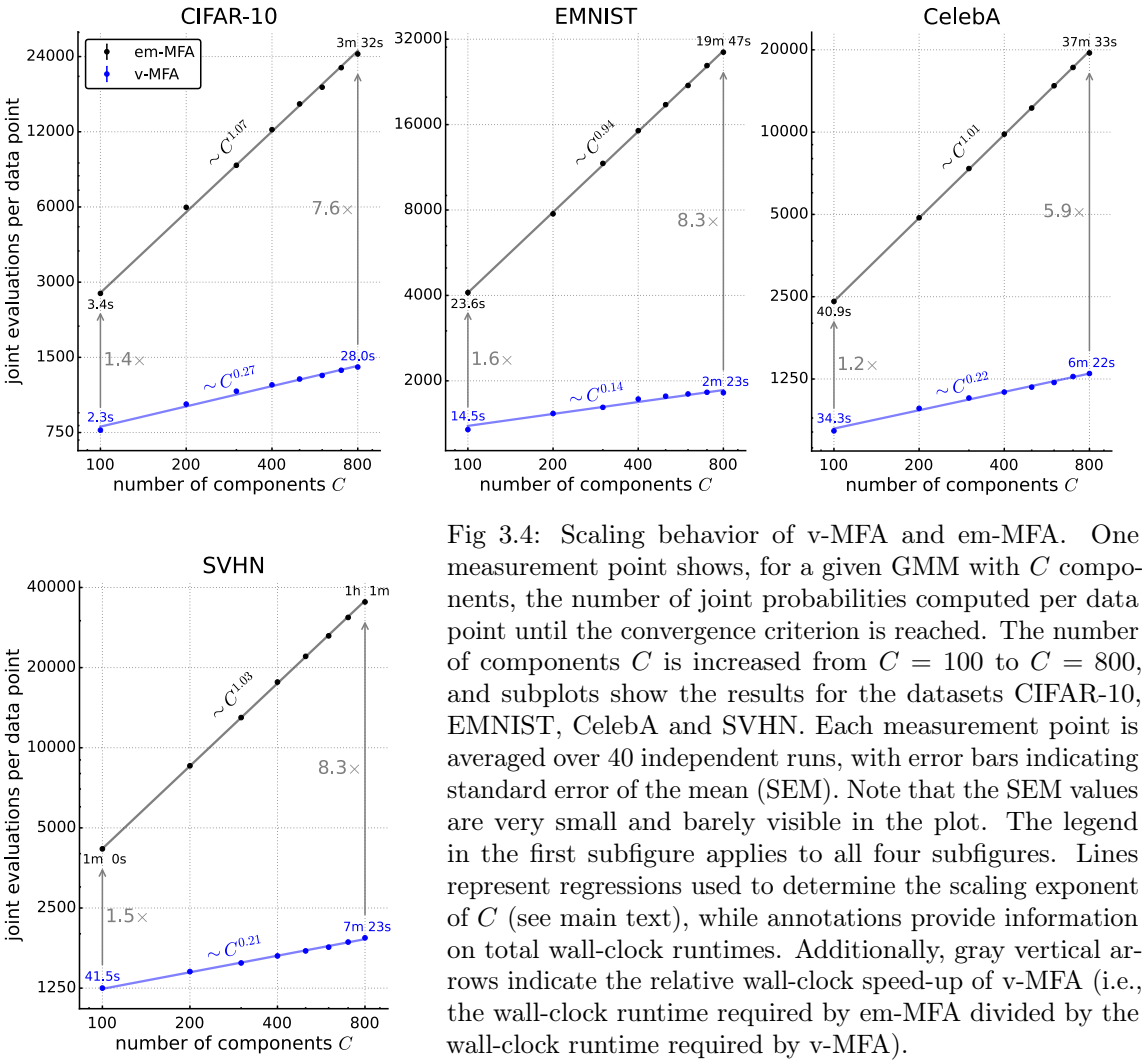

Fig 3.4: Scaling behavior of v-MFA and em-MFA. One measurement point shows, for a given GMM with C components, the number of joint probabilities computed per data point until the convergence criterion is reached. The number of components C is increased from $C = 100$ to $C = 800$, and subplots show the results for the datasets 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).

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. Sec. 3.2.3). 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 datasets. 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 Fig. 3.4), we discerned 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 3.3.3).

The results in Fig. 3.4 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.2.2.3). When we compare the computational demand required to achieve the same optimization quality (in terms of $\mathrm{NLL}_{\mathrm{test}}$), the speed-up of v-MFA over em-MFA remains essentially consistent with Fig. 3.4. 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.

3.3.2 Quality vs. Efficiency Analysis

In Sec. 3.3.1, 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 datasets.

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 testsets.

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.2.1.1). This approach follows Richardson and Weiss (2018), who used k-means and factor analysis for MFA initialization.

For the four datasets, 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 datasets. 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 3.5 shows the measured $\mathrm{NLL}_{\mathrm{test}}$ values vs. wall-clock runtimes for all algorithms. As can be observed, em-MFA achieved the best (lowest) NLL values on the EMNIST and SVHN datasets, while v-MFA shows the best NLL values on the CIFAR-10 and CelebA (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 baseline by approximately 1% for datasets such as CelebA and CIFAR-10, and up to 3% and 7% for the SVHN and EMNIST datasets, respectively.

In terms of runtime, v-MFA (with $C' = 3$ and $G = 5$) was the fastest among the algorithms evaluated on all four datasets (excluding Initialization). Across all settings of hyperparameters C' and G, v-MFA consistently outperformed em-MFA, achieving substantial speed-ups ranging from 3.3× to 25.6×. Additionally, v-MFA achieved faster runtimes than k-means+FA for the majority of hyperparameter settings.

Further details on the results of these experiments are provided in Appendix B.2.3. Additionally, we present in Appendix B.2.2.2 control experiments for v-MFA, em-MFA and k-means+FA compared to previously studied large-scale MFA optimization (Richardson and Weiss, 2018) that used a stochastic gradient descent (SGD) approach.

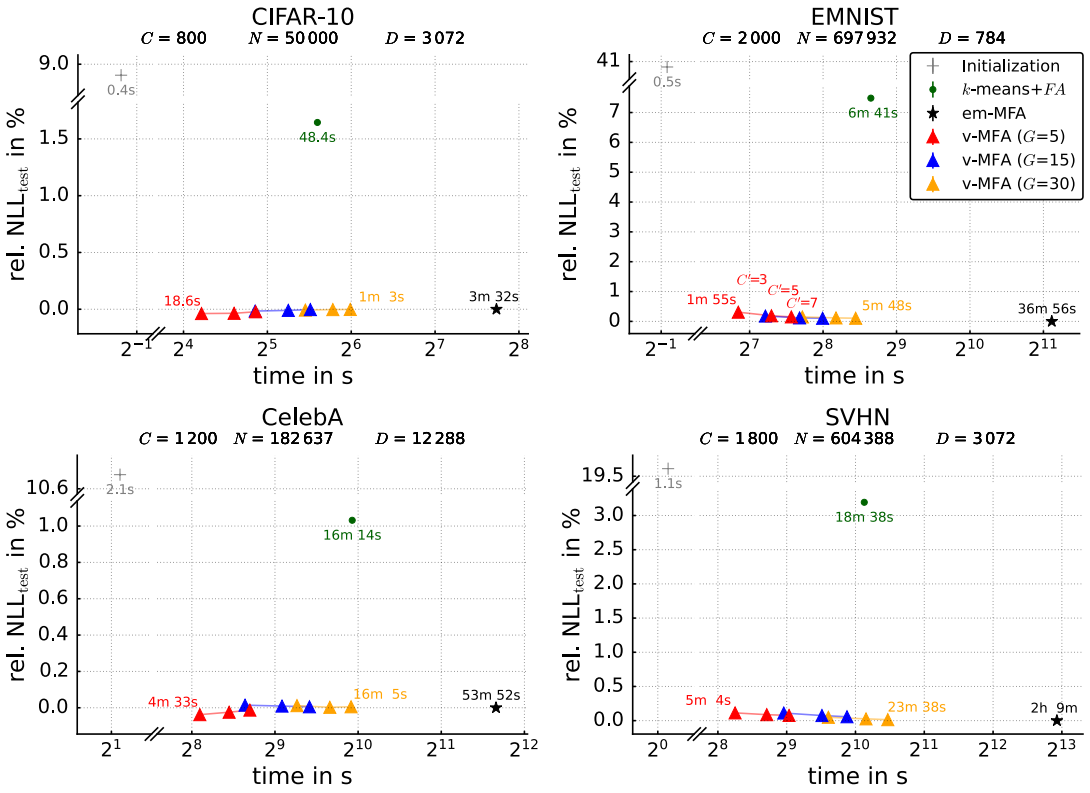

Fig 3.5: Effectiveness and efficiency in terms of negative log-likelihood (NLL) and runtime of v-MFA compared to em-MFA, k-means+FA and the parameters after initialization. Each of the four subfigures refers to experiments on one benchmark dataset. The y-axes denote the relative NLL on the testset w.r.t. em-MFA as baseline, given by Eq. (3.16). 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).

3.3.3 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 dataset of approximately 100M images, each sized $D = 32 \times 32 \times 3 = 3072$. The used dataset is a version of the YFCC100M dataset, with 5% of the samples randomly selected for testing and the remaining 95% for training (see Appendix B.2.1.3). We then applied v-MFA with increasingly many components. We used $C = 500$, 5000, 50 000, and 500 000. Unlike in Sec. 3.3.1, optimization by v-MFA used all training samples, regardless of the chosen value of C. The other hyperparameters were set as in Sec. 3.3.1, 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-MC^2) 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 $\boldsymbol{\mu}_{1:C}$, we selected data points uniformly at random (without duplicates). The remaining parameters were initialized as before. 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 10^{-3}, a larger value than in the previous smaller-scale experiments (see Sec. 3.2.4). 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 3.6. 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; Henighan *et al.*, 2020; Kaplan *et al.*, 2020; Wortsman *et al.*, 2024) 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.,

$$C + CD + CDH + CD = C(D(H + 2) + 1). \tag{3.17}$$

In Fig. 3.6**A**, 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 testset NLL (Fig. 3.6**B**). 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 Sec. 3.3.2, 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× as many joint evaluations). For $C = 5000$ the runtime of em-MFA would still be slower (by about an order of magnitude). Additionally, Fig. 3.6**A** 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[4], exceptionally large MFA models can be trained efficiently (see Fig. 3.6). Concretely, for $C = 500\,000$ an MFA model with 10.8 billion parameters can be optimized in less than nine hours.

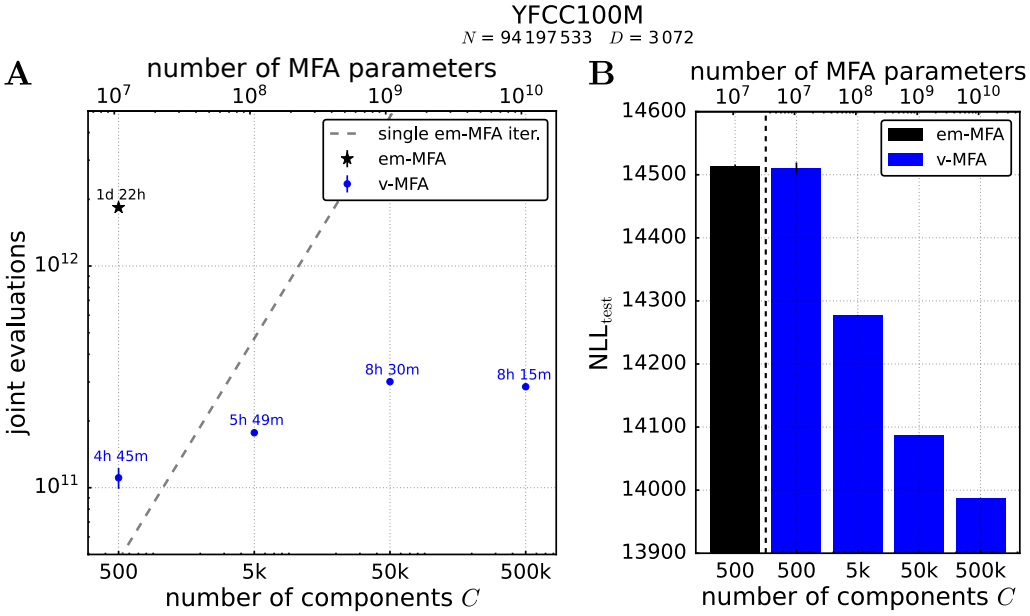

Fig 3.6: Evaluation of v-MFA and em-MFA on the YFCC100M dataset. **A** shows for different numbers of components C the total number of joint evaluations required for one v-MFA optimization run. The number of v-MFA model parameters is shown at the top. For each C we also provide the total runtime as annotation. 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 testset 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).

3.4 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., Li and Barron, 1999; Maz'ya and Schmidt, 1996; Zeevi and Meir, 1997), 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.

[4] 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.

Possible future research directions are the treatment of generalizations of MFAs towards non-Gaussian observation spaces including observables of different types (Banerjee *et al.*, 2005; Khan *et al.*, 2010), or extensions towards discrete time-sequence models (Rabiner, 1989; Zhou and Carin, 2015). Furthermore, future work may additionally make use of other learning strategies (e.g., Archambeau *et al.*, 2008; Elidan and Friedman, 2012; Lin *et al.*, 2019; Zhao and Yu, 2008), and it 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 (Ding and He, 2004; Hyvärinen, 2015).

In comparison to possible extensions, GMMs remain relatively elementary but they are also ubiquitous as standard metric data is very common. 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 are approximated 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 provides 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 Secs. 3.3.1 and 3.3.2). 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 Sec. 3.3.3). 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.

Future work using MFAs 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 deep generative models. 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.*, 2024; also compare Frey *et al.*, 2023; Hsu *et al.*, 2021, 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. Especially when considering that large datasets 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 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) generative models.

Chapter 4

Variational Graph-Based Semi-Supervised Learning

In this chapter, we build upon the variational optimization techniques introduced in Ch. 3 and integrate them into a recent graph-based semi-supervised learning (GSSL) framework. The integration alleviates the high computational costs typically associated with conventional graph-based optimization. Specifically, the GSSL framework is based on data distribution models and proceeds by first learning such a model, followed by graph construction and label propagation on the learned representations (compare Sec. 4.2.2). However, learning the data distribution itself often becomes a new bottleneck, limiting the scalability and efficiency of distribution-based GSSL methods on large, high-dimensional datasets (e.g., Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023). We address this bottleneck by integrating the variational techniques into a distribution-based GSSL framework (see Sec. 4.2.3). As distribution models, we use Gaussian mixture models (GMMs) with diagonal covariance matrices as well as mixtures of factor analyzers (MFAs), resulting in the v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL algorithms, respectively (cf. Sec. 4.2.4). In particular, the v-MFA algorithm introduced in Ch. 3 is integrated into v-MFA-GL.

This chapter is currently under review as a journal publication with authors Sebastian Salwig*, Till Kahlke* and Jörg Lücke (*joint first authorship) and title: Variational Graph-Based Semi-Supervised Learning. Below, we present the abstract of the paper, followed by a detailed description of the individual contributions made by each author. The remainder of this chapter corresponds to the sections of the paper. The associated appendix is provided in Appendix C. Appendix C.2.5 is not included in the paper and is exclusive to this doctoral thesis.

Abstract. Graph-based approaches provide well-established, versatile, and robust algorithms for semi-supervised learning. Recent advances in graph-based semi-supervised learning (GSSL) have addressed the scalability challenges for large modern datasets by leveraging data distribution models. However, the optimization of the distribution model then becomes the bottleneck for computational efficiency. Furthermore, the distribution model's quality and its interplay with graph-based labeling determines the effectiveness of the approach. To address efficiency and effectiveness, we derive a novel GSSL method which integrates variational techniques into a recent distribution-based GSSL framework. Concretely, we apply truncated variational approaches which scale sub-linearly with the

product of model and data size, and provide sparse graphs for efficient and effective GSSL. As distribution models, we use Gaussian mixture models (GMMs) and mixtures of factor analyzers (MFAs). We numerically evaluate the resulting GSSL algorithm on seven standard benchmarks. Using GMMs with diagonal covariances as distribution models, the algorithm improves the state-of-the-art in accuracy and substantially in runtime for all benchmarks. With MFAs as distribution models, the applied variational technique maintains fast runtimes while improving the state-of-the-art in accuracy in six of seven benchmarks. In many of these benchmarks, accuracies are improved by large margins.

Author Contributions. The idea of combining variational optimization techniques with graph-based semi-supervised learning was initially conceptualized by Sebastian Salwig (SS) and Jörg Lücke (JL). Preliminary numerical studies were conducted by SS with input from JL.

The literature review on semi-supervised learning was primarily carried out by SS with contribution from Till Kahlke (TK) and JL. The review of efficient optimization methods for distribution models was jointly carried out by all authors. The initial design of the algorithms v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL was developed by SS. Subsequent refinements included the incorporation of hot posteriors, proposed by TK, and the use of conjugate gradient, proposed by SS. The derivation of the probability product kernel for MiMoLaP was performed by SS and verified by TK. Dataset selection (excluding MNIST8M) was performed by SS, while MNIST8M was selected by TK, both in consultation with JL. TK and SS performed the numerical experiments with input from JL. Theoretical aspects and obtained results were throughout the study discussed among all authors. Figures and tables were created by TK and SS with input from JL. The manuscript was written by all authors. All authors approved the final version of the manuscript. The experiments and figures presented in subsection Appendix C.2.5 were carried out by SS, who also wrote the corresponding text.

The software was collaboratively developed by TK and SS. The implementation of comparison algorithms was done by SS with contributions by TK.

4.1 Introduction

Classification tasks are one of the core tasks in machine learning, and they are widely applied in various domains, such as medicine (Chebli *et al.*, 2018), speech recognition (Deng and Li, 2013), and document classification (Duarte and Berton, 2023). Supervised classifiers, e.g., Deep Neural Networks (DNNs), are typically the first choice when the dataset is fully labeled, as they often achieve the highest classification performance in such cases. While unlabeled data is usually relatively easy to obtain, labels are often scarce, as labeling can be a costly and time-consuming process (e.g., van Engelen and Hoos, 2020). As a result, it can be very challenging to infer the labels for data points if only a limited number of labels is available for training. The field of semi-supervised learning (SSL) addresses this issue: Given a dataset with a potentially small fraction of labeled data points, SSL aims to infer the labels of all unlabeled data points.

A fundamental assumption in semi-supervised learning is the *smoothness assumption*: any data points that are close to each other are likely to share the same label. Graph-based SSL (GSSL) methods, which are the focus of this work, inherently reflect this assumption through their graph structures (Song *et al.*, 2022). GSSL methods generally involve two

main steps: graph representation and label propagation. In the first step, data points are connected based on similarity measures. Thereby, the data points correspond to the nodes of the graph and the connections among the data points define the edges of the graph. The graph is typically represented by an adjacency matrix, which encodes the neighborhood relationships between the nodes. The second step involves propagating label information from the labeled nodes to the unlabeled nodes. This process is usually performed under some form of regularization to ensure that the propagated labels are consistent with the underlying structure of the graph.

While most conventional graph-based learning methods aim for high classification accuracies, their scalability with the number of data points is often disadvantageous. More concretely, for a dataset with N data points, graph construction in conventional GSSL methods typically scales with $\mathcal{O}(N^2)$, while the label propagation process typically scales with $\mathcal{O}(N^3)$. Such a scaling severely limits their applicability to the by now very common datasets with large N.

In order to address the limited scalability of conventional GSSL methods, several approaches have been proposed. One promising direction addresses the cubic-time complexity by first learning a data distribution model, followed by graph learning and labeling on the learned representations. These methods significantly reduce the super-linear complexity in N, which thus substantially accelerates graph construction and label propagation. However, distribution-based GSSL requires an optimized distribution model, and the corresponding optimization can itself become prohibitively time-consuming for large datasets. As a result, the optimization of the used distribution model often emerges as the new computational bottleneck of a distribution-based GSSL method.

A related challenge in GSSL is how the quality (i.e., accuracy) of the inferred labels can be improved. The accuracy depends on the process of graph construction and label propagation but (for distribution-based methods) also on the quality of the distribution model.

In our study, we contribute to the above stated challenges in GSSL as follows:

(A) We propose the first approach that integrates efficient variational optimization techniques (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) into a distribution-based GSSL framework, thereby reducing the high computational cost of distribution learning, graph construction and label propagation.

(B) We substantially enhance the accuracy of distribution-based GSSL by leveraging more flexible mixtures models in the form of mixtures of factor analyzers (MFAs; Salwig *et al.*, 2025).

For contribution (A), we utilize a variational optimization technique based on truncated posterior approximations which have previously been shown to enable highly efficient distribution learning at large-scales (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Salwig *et al.*, 2025). Here, we apply them for GSSL, which addresses the computational bottleneck of distribution learning. At the same time, the variational technique applied accelerates and improves graph learning and labeling. The accelerations for label propagation are due to the applied truncated posteriors as variational distributions, which inherently procures sparse graphs.

For contribution (B), we make use of GMMs that allow for a flexible modeling of covariances within their components. Concretely, we here integrate MFAs as distribution models into GSSL. MFAs allow for modeling correlations within components along low-dimensional

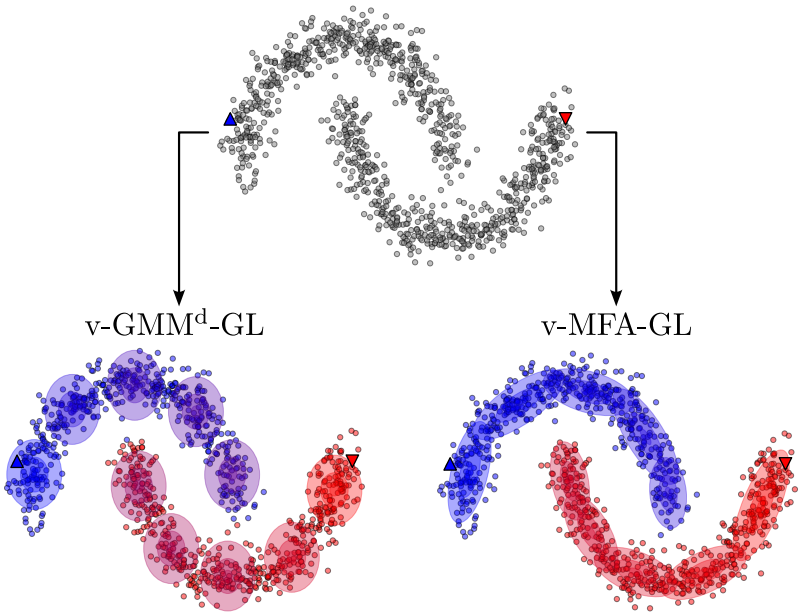

Fig 4.1: Illustration of distribution learning, graph learning, and labeling. At the top, a "two moons" dataset is displayed with two labeled points: a blue triangle on the left for one label, and a red triangle on the right for the other label. For distribution learning, v-GMM^{d}-GL (bottom left) uses a Gaussian mixture model (GMM) with diagonal covariances, shared among components; v-MFA-GL (bottom right) uses a mixture of factor analyzers (MFA) as distribution model. Both models use ten components, and each ellipse visualizes the covariances structure of one component. Instead of learning a graph with data points as nodes, the graph is constructed with mixture components as nodes. To infer soft class scores for each Gaussian component, label information is propagated between labeled data points and Gaussian components through the (soft) assignments of the distribution models, and label information is propagated across mixture components via the constructed graph (until a global state is reached). The color of each component (ranging form blue to red) indicates its soft class score. These soft class scores, together with the soft assignments between data points and Gaussian components, are finally used to predict the labels of the unlabeled data points (blue or red).

hyperplanes, which aligns with the manifold hypothesis assumed for many types of data. Previous scalable GSSL methods abstained from modeling correlations in their distribution models (and reasons related to memory efficiency and overfitting are stated; Zhang *et al.*, 2023).

As an illustration of the two here proposed GSSL methods, Figure 4.1 shows a GMM with diagonal covariances as distribution model and the more flexible MFA model (both fitted to the same dataset).

4.1.1 Related Work

Related work includes two areas of research: (graph-based) semi-supervised learning as well as the efficient optimization of distribution models. In the following, we will review the related work in these fields separately, beginning with semi-supervised learning.

GSSL: The literature on semi-supervised learning and, more specifically, graph-based SSL is extensive and has evolved significantly over time. For details on recent advancements, we refer to two recent overview papers (Song *et al.*, 2022, 2023).

A pioneering contribution to GSSL was made by Zhu *et al.* (2003), who introduced a framework leveraging Gaussian random fields to address semi-supervised learning problems on graphs. Building on this, Belkin and Niyogi (2004) proposed defining a classification function restricted to the data submanifold, rather than over the entire ambient space. Subsequently, Zhou *et al.* (2003) presented a classification framework designed to maintain sufficient smoothness aligned with the intrinsic structure formed by both labeled and unlabeled data points. Such conventional GSSL approaches share the common characteristic of operating directly on the dataset. Therefore, a major limitation of the conventional GSSL methods is their limited scalability when applied to large datasets. Here, we focus on a subset of methods that share a similar approach, designed to enhance the efficiency of conventional GSSL methods. These methods first learn the data density distribution, followed by graph construction and graph regularization on the learned representation.

Landmark or *anchor* based methods can bee seen as such approaches which employ the k-means algorithm to find a small set of anchor points and construct a graph based on these points. Among the first of these methods is the Anchor Graph Regularization (AGR), proposed by Liu *et al.* (2010). The primary advantage of this method is a linear scaling with respect to N. An enhanced version, called Efficient Anchor Graph Regularization (EAGR), was introduced by Wang *et al.* (2016) and builds upon the AGR framework. EAGR addresses the limitations in both the anchor graph construction and the anchor graph regularization steps. Another method building upon AGR was proposed by Wang *et al.* (2017). This hierarchical AGR leverages a pyramid-style structure with multiple-layer anchors to construct the adjacency matrix. Furthermore, Qiu *et al.* (2019) proposed to combine anchor-based graphs (Liu *et al.*, 2010; Wang *et al.*, 2016) with a flexible manifold embedding (FME) technique (Nie *et al.*, 2010). The resulting two approaches were referred to as fast Flexible Manifold Embedding (f-FME) and reduced Flexible Manifold Embedding (r-FME), respectively.

In contrast to the previously discussed anchor-based methods using k-means, Zhu and Lafferty (2005) were among the first to employ a probabilistic distribution model in the form of a Gaussian mixture model (GMM) in order to reduce the original graph to a smaller 'backbone' graph. However, this approach still necessitates the construction of an $N \times N$ graph matrix over the data, leading to a computational complexity of $\mathcal{O}(N^2)$, which limits its scalability to large datasets. Building on this work, Li *et al.* (2020) proposed an approach in which the graph is constructed based on the similarities between mixture components. This strategy effectively eliminates the dependency of the time complexity on the size of the dataset in the graph representation and label propagation step. Another approach utilizing GMMs, called Mixture Model Label Propagation (MiMoLaP), was introduced by Chi *et al.* (2010). In this method, the complexity of graph construction is also independent of N, as the graph is constructed using the Probability Product Kernel (PPK) between the Gaussian mixture components. Finally, Zhang *et al.* (2023) adopted a classification framework similar to AGR and EAGR, but learned the data distribution using GMMs, naming their method Data Distribution base Graph Learning (DDGL).

Efficient Optimization of Distribution Models: Rendering the optimization of data distribution models more efficient is a general challenge. Several directions to make, e.g., k-means-like approaches or GMMs more efficient have been explored. The different directions aim to reduce the scalability, for instance, w.r.t. the number of data points N, the number of components C, the dimensionality D, or they aim at reducing the number of iterations required for convergence. To reduce scaling with N, approaches such as coresets

(Lucic *et al.*, 2018), mini-batching (Nguyen *et al.*, 2020), or training on separate subsets of the dataset (Liu *et al.*, 2024) have been proposed. Approaches aimed at reducing the scaling with D include geometrically-oriented methods Elkan (2003), random projections (Chan and Leung, 2017), and dimensionality reduction techniques (Ghojogh *et al.*, 2023; Hertrich *et al.*, 2022; Liu *et al.*, 2022; Richardson and Weiss, 2018). Seeding methods (Arthur and Vassilvitskii, 2007; Bachem *et al.*, 2016a) can reduce the number of iterations by providing well-chosen initial parameters.

More relevant to this work, however, are approaches that address the scaling with the number of components C. In general, variational methods are a widely used technique for efficient optimization of generative models (Jordan *et al.*, 1999; Neal and Hinton, 1998). They are applicable to models with discrete latents (e.g., Drefs *et al.*, 2022; Zhang *et al.*, 2019) and, in particular, they were used to accelerate the optimization of mixture models (e.g., Dai and Lücke, 2014; Forster *et al.*, 2018; Neal and Hinton, 1998; Shelton *et al.*, 2017). For GMMs constraint to diagonal covariance matrices, recent contributions (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) have demonstrated that variational optimization reduces the complexity class for parameter optimization from scaling linearly with NC to a sublinear scaling. Still more recently (Salwig *et al.*, 2025), it was shown that GMMs with highly flexible covariance matrices can be optimized with similar efficiency. By combining factor analysis (cf. Ghahramani and Hinton, 1996; Richardson and Weiss, 2018) per mixture component with variational techniques, runtime complexities can be achieved that scale linearly with D and sublinearly with NC.

4.2 Methods

4.2.1 Preliminaries

We address a standard SSL problem for multi-class classification in which typically only a small fraction of the data is labeled (Fig. 4.1 shows an extreme case of just one labeled data point per class). Let the set of data points be represented by $\mathcal{X} = \boldsymbol{x}_{1:N}$, where each data point $\boldsymbol{x}_n$ is a point in $\mathbb{R}^D$. Correspondingly, let the set of labels be denoted as $\mathcal{Y} = y_{1:N_l}$, where the first N_l data points are labeled, and each label y_n belongs to one of K discrete classes, i.e., $y_n \in \{1, ..., K\}$. The remaining $N_u = N - N_l$ data points are unlabeled.

The here addressed standard SSL task is to infer the labels of the unlabeled data points based on the data points $\mathcal{X}$ in $\mathbb{R}^D$ and the label set $\mathcal{Y}$. Graph-based SSL algorithms address this task by first constructing a weighted graph using the dataset $\mathcal{X}$. Such a graph can be described by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, where $\mathcal{V}$ denotes the nodes, $\mathcal{E}$ denotes the edges of the graph and $\boldsymbol{W}$ is an adjacency matrix indicating the weights of the edges. In conventional GSSL methods, the data points themselves are used as nodes in the graph (both unlabeled and labeled data points). The edges between the nodes are assigned weights based on similarity measures, often computed using distance metrics, such as a Gaussian kernel (e.g., Zhu and Lafferty, 2005). The graph can be either fully or sparsely connected, employing methods such as k-NN (Eppstein *et al.*, 1997). Once the graph is built, it is used to propagate label information from the labeled nodes to the unlabeled ones. This process is typically performed under some regularization framework (e.g., Zhou *et al.*, 2003). A GSSL method usually seeks to find a classifying function that satisfies two key criteria: (A) The classifying function should not deviate significantly from the initial label assignments for the labeled data points (*fitting constraint*). (B) The classifying function should vary smoothly between

adjacent nodes in the graph, meaning that data points connected by high-weight edges (indicating similarity) should have similar label predictions (*smoothness constraint*).

A significant limitation of conventional GSSL methods, which operate directly on the data points, is their unfavorable scaling of computational demand with increasing dataset sizes. While the computational complexity of constructing the graph scales with $\mathcal{O}(N^2)$, the complexity of the graph regularization step grows with $\mathcal{O}(N^3)$, due to the computation of the inverse of the graph kernel.

4.2.2 Graph Learning based on Data Distributions

In order to prevent the above stated unfavorable scaling with the number of data points N, a prominent line of research for GSSL involves first learning a data representation using a parametric data distribution model. Such a learned representation is then used for graph construction and label propagation (compare related works in Sec. 4.1.1). Instead of directly constructing a graph on the data points, these methods construct the graph based on learned data patterns. As a result, they reduce the computational complexity from scaling cubically with N to linear scaling. Distribution-based methods do, however, introduce an additional step of learning the data representation, which involves, e.g., the application of the k-means algorithm (e.g., Wang *et al.*, 2016) or the optimization of GMMs (e.g., Zhang *et al.*, 2023). In this context, k-means may be interpreted as an approximated and constrained GMM optimization (Lücke and Forster, 2019). The conventional approach for optimizing such models is the expectation maximization (EM) algorithm (Dempster *et al.*, 1977), which typically scales with a computational complexity of $\mathcal{O}(NCD)$ per iteration, where C represents the number of clusters or components, and D denotes the dimensionality of the data[5].

It can be expected that distribution models that can parameterize the underlying data distribution more flexibly enhance the performance of GSSL methods compared to more rigid models. For instance, results in Li *et al.* (2020) and Zhang *et al.* (2023) show that employing a GMM results in better accuracies compared to GSSL methods that construct the graph on data representations obtained by simpler models, such as constrained and approximate GMM models implied by the application of the k-means algorithm. However, optimizing a GMM is generally more time-intensive than optimization using k-means, and for increasingly general GMMs (or more complex models) the computational cost can be expected to further increase.

Consequently, learning the data distribution can become significantly time-consuming, particularly when applied to large-scale datasets. In fact, the distribution learning step often is the computationally most expensive part of a GSSL algorithm (cf. Li *et al.*, 2020; Wang *et al.*, 2016).

4.2.3 Variationally Accelerated Distribution and Graph Learning

To further enhance the efficiency of GSSL methods, we propose a novel approach that addresses the bottleneck in efficiency and effectiveness of distribution-based GSSL. Concretely, our method integrates variational optimization techniques for GMMs (Hirschberger *et al.*, 2022; Salwig *et al.*, 2025) into the framework of recent GSSL methods (Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023), which construct their graphs on learned data

[5] The conventional optimization of GMMs with full covariance matrices scales with $\mathcal{O}(NCD^2)$.

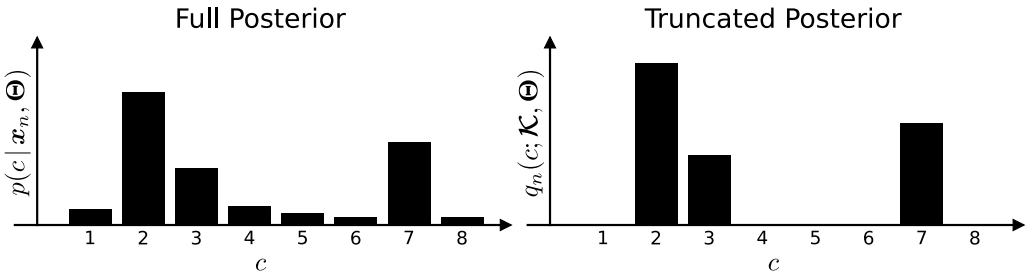

Fig 4.2: Full and truncated posterior distributions. Left: For a given data point $\boldsymbol{x}_n$, the majority of components typically exhibit small posterior values. Right: By selecting $\mathcal{K}^{(n)} = \{2, 3, 7\}$, the truncated posterior captures most of the probability mass of the full posterior.

representations. To model the underlying data distributions, we explore the application of (constrained) GMMs as distribution models as well as MFAs, which will be introduced in more detail in Sec. 4.2.4.

In the following, we first discuss the data distribution learning step based on the variational optimization techniques for GMMs and then continue explaining how these techniques simultaneously accelerate graph learning and label propagation.

Distribution Learning: To accelerate the data distribution learning step, a variational version of the EM algorithm (Hirschberger *et al.*, 2022; Salwig *et al.*, 2025) is used, which itself is based on truncated posteriors as its family of variational distributions. Truncated posterior distributions are given by

$$q_n(c; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}) = \frac{p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta})}{\sum_{c' \in \mathcal{K}^{(n)}} p(c', \boldsymbol{x}_n \mid \boldsymbol{\Theta})} \delta(c \in \mathcal{K}^{(n)}), \tag{4.1}$$

where $\delta(c \in \mathcal{K}^{(n)})$ equals one if $c \in \mathcal{K}^{(n)}$ and zero otherwise (we use δ to denote the Iverson bracket), $\mathcal{K}^{(n)}$ represents the set of component indices for which the variational distribution $q_n(c; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$ is non-zero and $\boldsymbol{\mathcal{K}}$ is the collection of all index sets $\mathcal{K}^{(n)}$. The size of the sets $\mathcal{K}^{(n)}$ is restricted to $C' \leq C$ indices for each data point. The truncated posteriors, in comparison to the full posteriors, are illustrated in Fig. 4.2. The objective to optimize in the variational EM version used here is the free energy (also known as the Evidence Lower Bound - ELBO), which, when truncated posteriors are used, is given by:

$$\mathcal{F}(\boldsymbol{x}_{1:N}; \boldsymbol{\mathcal{K}}, \tilde{\boldsymbol{\Theta}}, \boldsymbol{\Theta}) = \sum_{n,c} q_n(c; \boldsymbol{\mathcal{K}}, \tilde{\boldsymbol{\Theta}}) \log p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta}) + \mathcal{H}\left[q_n\right], \tag{4.2}$$

where $\mathcal{H}\left[q_n\right] = -\sum_n \mathbb{E}_{q_n}[\log q_n(c; \tilde{\boldsymbol{\Theta}})]$ is the Shannon entropy. The use of 'hard' zeros in the truncated distributions ensures that, in summations over components C, only C' non-zero terms need to be evaluated. As a result, the computational complexity of a single M-step is reduced from linear scaling with C to linear scaling with C'.

To maximize the free energy, the optimal variational parameters $\mathcal{K}^{(n)}$ have to contain the C' components with the highest joint probabilities, which correspond to performing a full E-step. To further reduce the computational complexity of the E-step, the here applied variational EM versions use partial variational E-steps (Hirschberger *et al.*, 2022; Salwig *et al.*, 2025), which involve evaluating only subsets of all joint probabilities. More concretely, search spaces $\mathcal{S}^{(n)}$ are used, which contain candidate components $\tilde{c}$ that may replace a

component $c \in \mathcal{K}^{(n)}$ to increase the free energy. Given the set $\mathcal{S}^{(n)}$, the $\mathcal{K}^{(n)}$ set is updated as follows:

$$\mathcal{K}^{(n)} = \{c \mid p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta}) \text{ is among the } C' \text{ largest joints } \text{ for all } c \in \mathcal{S}^{(n)}\}. \tag{4.3}$$

The size of $\mathcal{S}^{(n)}$ is upper-bounded by S. Thereby, S will be selected to be greater than C' but significantly smaller than C (i.e., $C' < S \ll C$). To include replacement candidates in the search spaces that are likely to increase the free energy, the search spaces contain components that have been estimated as 'similar' to the components already present in the $\mathcal{K}^{(n)}$ sets (for further details, we refer to Salwig *et al.*, 2025).

By using this variational EM approach, the computational complexity w.r.t. C is reduced from linear to constant per iteration, compared to the conventional EM algorithm.

Graph Learning and Labeling: For the graph learning and labeling stage, we build upon a GSSL framework that has been recently employed in several studies (Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023). The first step entails the construction of a transformation matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times C}$, which represents relationships between the data points and the mixture components. Therefore, we use the variational distributions and construct $\boldsymbol{Z}$ as follows:

$$Z_{nc} = q_n(c; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}). \tag{4.4}$$

Due to the specific choice of variational distributions (Eq. (4.1)), the transformation matrix $\boldsymbol{Z}$ in Eq. (4.4) inherits a sparse graph from the truncation of full posteriors to approximate posteriors with C' non-zero entries (i.e., $\boldsymbol{Z}$ has only C' non-zero entries per row). No extra sparsification step (as, e.g., applied by Liu *et al.*, 2010; Qiu *et al.*, 2019; Wang *et al.*, 2016) is required.

Based on the transformation matrix $\boldsymbol{Z}$, a graph is then constructed on the mixture components, and the weights of this graph $\boldsymbol{W} \in \mathbb{R}^{C \times C}$ are computed by

$$\boldsymbol{W} = \boldsymbol{Z}^T \boldsymbol{Z}, \tag{4.5}$$

where the diagonal elements of $\boldsymbol{W}$ are set to zero. To propagate the label information between the mixture components, a regularization framework is defined on a cost function. This function is conceptually similar to those used in previous work (Wang *et al.*, 2016; Zhang *et al.*, 2023; Zhou *et al.*, 2003), but it differs in the properties of the matrices used to define it. Furthermore, we use novel techniques to optimize the cost function within this framework. The cost function is given by:

$$\begin{aligned} Q(\boldsymbol{A}) &= \sum_{n=1}^{N_l} \left\| \boldsymbol{z}_n^T \boldsymbol{A} - \boldsymbol{y}_n \right\|^2 + \frac{\gamma}{2} \sum_{c,\tilde{c}=1}^{C} W_{c\tilde{c}} \left\| \frac{\boldsymbol{a}_c}{\sqrt{D_{cc}}} - \frac{\boldsymbol{a}_{\tilde{c}}}{\sqrt{D_{\tilde{c}\tilde{c}}}} \right\|^2 \\ &= \|\boldsymbol{Z}_l \boldsymbol{A} - \boldsymbol{Y}_l\|_F^2 + \gamma \operatorname{tr}\left(\boldsymbol{A}^T \boldsymbol{L} \boldsymbol{A}\right), \end{aligned} \tag{4.6}$$

where $\boldsymbol{A} \in \mathbb{R}^{C \times K}$ is a soft class matrix, which assigns a score from mixture component c to label class k, and $\boldsymbol{a}_c$ denotes the c-th row of $\boldsymbol{A}$. $\boldsymbol{Y}_l \in \mathbb{R}^{N_l \times K}$ is a class indicator matrix on labeled data points given by

$$(\boldsymbol{Y}_l)_{nk} = \begin{cases} 1 & \text{if} \quad y_n = k \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

Furthermore, $\boldsymbol{Z}_l \in \mathbb{R}^{N_l \times C}$ is the transformation matrix of the labeled data points. $\boldsymbol{L}$ is the normalized Laplacian matrix and $\gamma > 0$ is a regularization parameter that weights the fitting/smoothness trade-off. The normalized Laplacian matrix is given by

$$\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{W} \boldsymbol{D}^{-\frac{1}{2}}, \tag{4.8}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{D} \in \mathbb{R}^{C \times C}$ is a diagonal matrix with $D_{cc} = \sum_{\tilde{c}} W_{c\tilde{c}}$. Due to the sparsity of $\boldsymbol{Z}$, the weight matrix $\boldsymbol{W}$ in Eq. (4.5) and, consequently, the Laplacian matrix $\boldsymbol{L}$ in Eq. (4.8) can be computed very efficiently.

The regularization framework in Eq. (4.6) is based on the aforementioned principles, where the first term accounts for the fitting constraint, and the second term accounts for the smoothness constraint.

To obtain the optimal soft class matrix $\boldsymbol{A}$, Eq. (4.6) is minimized, resulting in a closed-form solution for $\boldsymbol{A}$, which is given by

$$\boldsymbol{A} = \left(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L}\right)^{-1} \boldsymbol{Z}_l^T \boldsymbol{Y}. \tag{4.9}$$

In previous work (Wang *et al.*, 2016; Zhang *et al.*, 2023), Eq. (4.9) has been applied to obtain the optimal soft class matrix $\boldsymbol{A}$, which involves the inversion of the extended Laplacian $(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L})$. However, due to the sparsity of $\boldsymbol{Z}$ as defined in Eq. (4.4), the extended Laplacian $(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L})$ in Eq. (4.9) also becomes sparse. Unfortunately, directly estimating the matrix $\boldsymbol{A}$ by Eq. (4.9) nevertheless remains impractical for large C because the resulting inverse matrix may not be sparse. Since the extended Laplacian $(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L})$ is positive-semidefinite, we use here the conjugate gradient (CG; Shewchuk, 1994) method to obtain $\boldsymbol{A}$ as the solution of the linear system

$$\left(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L}\right) \boldsymbol{A} = \boldsymbol{Z}_l^T \boldsymbol{Y}. \tag{4.10}$$

The efficiency of the CG method does not depend on the size of the extended Laplacian matrix $(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L})$, but rather on the number of its nonzero entries. Consequently, the CG method is expected to be more efficient than matrix inversion when applied to large, sparse extended Laplacian matrices. Finally, the labels of unlabeled data points are predicted as follows:

$$y_n = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \frac{(\boldsymbol{Z}\boldsymbol{A})_{nk}}{\lambda_k}, \quad \forall n \in N_l + 1, ..., N \tag{4.11}$$

where λ_k is given by the sum of the entries of the k-th column of $\boldsymbol{Z}\boldsymbol{A}$. The normalization with λ_k address the imbalance in skewed class distributions, following the method outlined in Zhu *et al.* (2003), Liu *et al.* (2010) and Wang *et al.* (2016).

4.2.4 GMMs for Data Distribution Learning

Having introduced our novel GSSL method in Sec. 4.2.3, we now focus on the effective estimation of underlying data distributions. In this work, we utilize two different types of GMMs. One well-known property of GMMs is their ability to approximate data densities arbitrarily well with increasing size (e.g., Li and Barron, 1999; Maz'ya and Schmidt, 1996; Zeevi and Meir, 1997). While such a universal probability density approximation is largely independent of the used type of GMM, the scaling of approximation quality with GMM size strongly varies with the used GMM type.

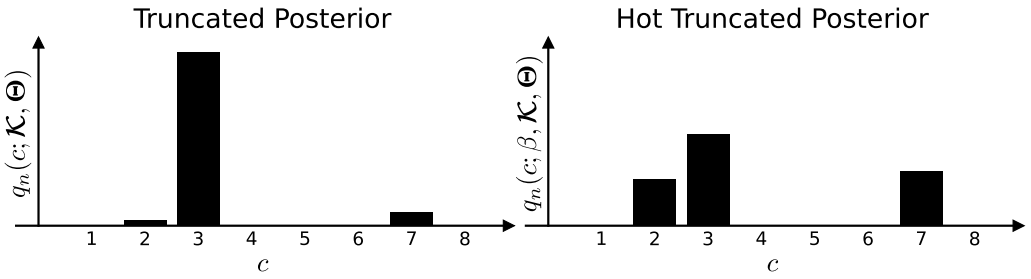

Fig 4.3: Truncated and hot truncated posterior distributions. Left: In this example, the majority of the probability mass is concentrated in a single component. Right: By applying an inverse temperature $\beta < 1$, the hot truncated posterior distribution becomes more uniform relative to the truncated posterior.

Building upon prior work (Zhang *et al.*, 2023), we first employ GMMs with diagonal and shared covariances. The generative model is defined as:

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{S}), \tag{4.12}$$

where $\pi_c \in [0, 1]$ are the mixing proportions, $\boldsymbol{\mu}_c \in \mathbb{R}^D$ are the means and $\boldsymbol{S} := \text{diag}(\sigma_1^2, ..., \sigma_D^2) \in \mathbb{R}^{D \times D}$ is a diagonal covariance matrix. We refer to the integration of the novel GSSL approach with GMMs with diagonal and shared covariances as v-GMM$^{\text{d}}$-GL.

However, constraining the covariance matrices to be diagonal reduces the flexibility of the GMMs, which can significantly hinder their ability to efficiently approximate data densities. Consequently, the overall performance of v-GMM$^{\text{d}}$-GL may be compromised.

Therefore, as an alternative, we employ more general GMMs, which provide the important ability to capture correlations within components. Concretely, we combine our novel GSSL approach with MFA models (Ghahramani and Hinton, 1996; Richardson and Weiss, 2018), and refer to the resulting algorithm as v-MFA-GL. The MFA model enables flexible modeling of correlations for each component along lower-dimensional hyperplanes. In addition to a hidden mixture component $c \in \{1, \ldots, C\}$, a data point is modelled by a hidden factor $\boldsymbol{z} \in \mathbb{R}^H$. The generative model of MFA is given by

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{z} \,|\, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}), \qquad p(\boldsymbol{x} \,|\, \boldsymbol{z}, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c, \boldsymbol{S}), \tag{4.13}$$

where $\boldsymbol{S}$ denotes a diagonal, shared covariance matrix (as in Eq. (4.12)), and $\boldsymbol{\Lambda}_c \in \mathbb{R}^{D \times H}$ is the factor loading matrix. The MFA model aligns well with the commonly made assumption that high-dimensional data lie close to a low-dimensional manifold, because it provides piecewise linear subspaces that are able to approximate low-dimensional manifolds. By marginalization $p(\boldsymbol{x} \,|\, \boldsymbol{z}, c, \boldsymbol{\Theta})$ over $\boldsymbol{z}$, the MFA model can equivalently be formulated as a GMM with low-rank plus diagonal covariance matrices, given by $\boldsymbol{\Sigma}_c = \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^T + \boldsymbol{S}$.

In Salwig *et al.* (2025), the variational EM framework was extended to general GMMs (including MFAs), facilitating highly efficient training for large-scale MFA models.

4.2.5 Avoiding Extremely Sparse Graphs

If the (truncated) posterior probabilities for the majority of the data points tend to approach one for a single component, while becoming increasingly small for the remaining components (see Fig. 4.3, top), the transformation matrix $\boldsymbol{Z}$ becomes overly sparse. As a result, the non-diagonal weights in the weight matrix $\boldsymbol{W}$ (as defined in Eq. (4.5)) approach zero and

information about the relationships between components may be compromised or lost. To prevent very small values in $\boldsymbol{Z}$, we 'heat up' the posteriors.

Concretely, we construct the transformation matrix $\boldsymbol{Z}$ using hot truncated posterior distributions as follows

$$Z_{nc} = q_n(c; \beta, \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta}) = \frac{p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})^\beta}{\sum_{c' \in \mathcal{K}^{(n)}} p(c', \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})^\beta} \delta(c \in \mathcal{K}^{(n)}), \tag{4.14}$$

where β can be seen as an inverse temperature. For $\beta < 1$, the non-zero entries of $\boldsymbol{Z}$ become more uniform (see Fig. 4.3 for an illustration). For v-GMM$^\mathrm{d}$-GL we used a fixed $\beta = 1/\sqrt{D}$, and for v-MFA-GL we used a fixed $\beta = 1/D$ across all experiments. We do not apply the hot posteriors *during* distribution learning, i.e., we use the truncated posteriors in Eq. (4.1) for the variational EM algorithm.

The resulting v-GMM$^\mathrm{d}$-GL and v-MFA-GL algorithms are summarized in Alg. 3.

Algorithm 3: v-GMM$^\mathrm{d}$-GL or v-MFA-GL

Distribution learning:

1: Train a GMM with diagonal and shared covariances (v-GMM$^\mathrm{d}$-GL) or MFA (v-MFA-GL) using variational EM.

Graph learning and labeling:

2: Set $Z_{nc} = q_n(c; \beta, \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})$.

3: Compute $\boldsymbol{W} = \boldsymbol{Z}^T \boldsymbol{Z}$ (with zero diagonal).

4: Compute $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{W} \boldsymbol{D}^{-\frac{1}{2}}$.

5: Solve $(\boldsymbol{Z}_l^T \boldsymbol{Z}_l + \gamma \boldsymbol{L}) \boldsymbol{A} = \boldsymbol{Z}_l^T \boldsymbol{Y}$ w.r.t. $\boldsymbol{A}$ using CG.

6: Predict the labels using Eq. (4.11).

4.2.6 Complexity Analysis

In this section, we analyze the computational complexity of the proposed algorithms and compare it with several state-of-the-art methods from the literature. The complexity analysis is categorized into two stages: the data distribution learning stage and the graph learning and labeling stage. The runtime complexities of the proposed algorithms, along with those of the comparative methods, are summarized in Tab. 4.1.

Tab 4.1: Time complexities (worst case) of several GSSL methods. MFA-GL is a non-variational version of v-MFA-GL (for more details, see Sec. 4.3). T denotes the number of iterations.

Method	Distribution Learning (per Iteration)	Graph Learning and Labeling
conventional GSSL	–	$\mathcal{O}(N^3)$
AGR	|	$\mathcal{O}(NCC' + NC'^2DT + C^3)$
EAGR		$\mathcal{O}(NCC' + NC'^2D + C^3)$
f-FME	$\mathcal{O}(NCD)$	$\mathcal{O}(N(C+D+1)^2 + (C+D+1)^3)$
r-FME		$\mathcal{O}(NC^2 + C^3 + C^2D + CD^2 + D^3)$
MiMoLaP	|	$\mathcal{O}(C^2D + C^3)$
DDGL		$\mathcal{O}(NC^2)$
MFA-GL	$\mathcal{O}(NCDH)$	$\mathcal{O}(NC^2)$
v-GMM$^\mathrm{d}$-GL	$\mathcal{O}(NSD)$	$\mathcal{O}(NC'^2 + C^3)$
v-MFA-GL	$\mathcal{O}(NSDH)$	$\mathcal{O}(NC'^2 + C^3)$

The computational complexity of previous distribution-based GSSL methods is equal in the distribution learning stage. But the methods have different computational complexities in their corresponding graph learning and labeling stages.

Among the approaches compared in Tab. 4.1, MiMoLaP and the here proposed algorithms v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL achieve the lowest time complexities in the graph learning and labeling stage, depending on the values of N, C, C', and D. However, the overall time complexities of all algorithms are dominated by the distribution learning stage, which also scales with the number of iterations required for optimization. In this first stage, the lowest time complexities are achieved by v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL. The use of variational EM optimization reduces the dependency on C to S, with $S \ll C$. For diagonal GMMs, the complexity per iteration decreases from $\mathcal{O}(NCD)$ (conventional EM) to $\mathcal{O}(NSD)$ (Hirschberger *et al.*, 2022), and for MFAs, it decreases from $\mathcal{O}(NCDH)$ to $\mathcal{O}(NSDH)$ (Salwig *et al.*, 2025).

Note that for AGR and EAGR, the original papers (Liu *et al.*, 2010; Wang *et al.*, 2016), along with other studies (Zhang *et al.*, 2023), reported a dominating term of order $\mathcal{O}(NC^2)$ for the graph learning and labeling stage, which is attributed to the computation of the Laplacian and holds true in the general case. However, by exploiting the specific sparse structure inherent in AGR and EAGR (similar to that of our method), we note that the computation can be performed in $\mathcal{O}(NC'^2)$. Here, C' is the numbers of anchors associated with each data point, i.e., the number of non-zero entries in each row of $\boldsymbol{Z}$ (denoted as s in Liu *et al.*, 2010; Wang *et al.*, 2016).

4.3 Results

To evaluate the efficiency and effectiveness of the v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL algorithms, we conduct a series of numerical experiments, structured into three sections. In Sec. 4.3.1, we evaluate the runtimes of our proposed variational algorithms in comparison with several established as well as recent GSSL algorithms. Following the runtime analysis, we benchmark in Sec. 4.3.2 the performance and quality of inferred labels when different numbers of labeled data points are provided. Finally, in Sec. 4.3.3, we extend the performance and quality analysis to an extreme case scenario with only a single labeled data point per class.

For consistency with one of the most recent GSSL contributions (Zhang *et al.*, 2023), we evaluate all algorithms (the proposed and others) on six well-established benchmarks, namely USPS (Hull, 1994), Letter (Slate, 1991), CIFAR-10 (Krizhevsky *et al.*, 2014), MNIST (Lecun *et al.*, 1998), EMNIST (Cohen *et al.*, 2017) and SUSY (Baldi *et al.*, 2014), and apply similar preprocessing pipelines. Furthermore, we include MNIST8M (Loosli *et al.*, 2007) as an additional large-scale benchmark. The main properties of the datasets and the used range for labeled data points N_l are summarized in Tab. 4.2. Additional details on both the datasets and preprocessing are provided in Appendices C.1.1 and C.1.2.

On all benchmarks, we numerically evaluate the performance of the here derived algorithms, v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL. We also evaluate a third novel algorithm, termed MFA-GL. Like v-MFA-GL, the MFA-GL algorithm uses MFA as distribution model, which has previously not been used for GSSL. However, unlike v-MFA-GL, the MFA-GL approach does not use a variational acceleration but conventional EM is applied instead. Consequently, the transformation matrix $\boldsymbol{Z}$ in Eq. (4.4) is constructed using the full posterior distributions instead of truncated posteriors, and, furthermore, no hot posteriors (as defined in Eq. (4.14))

Tab 4.2: Discretion of the used datasets. For D, the numbers in parentheses denote the data dimension after preprocessing. For Letter and SUSY, no preprocessing is applied.

	Dataset	N	D	K	N_l
USPS	Hull (1994)	9298	256 (30)	10	$30 - 100$
Letter	Slate (1991)	20 000	16	26	$100 - 500$
CIFAR-10	Krizhevsky (2009)	60 000	3096 (30)	10	$100 - 500$
MNIST	Lecun *et al.* (1998)	70 000	784 (30)	10	$100 - 500$
EMNIST	Cohen *et al.* (2017)	280 000	784 (30)	10	$50 - 200$
SUSY	Baldi *et al.* (2014)	5 000 000	18	2	$20 - 100$
MNIST8M	Loosli *et al.* (2007)	8 100 000	784 (30)	10	$50 - 500$

are employed. We do not include an additional non-variational version of v-GMM$^{\mathrm{d}}$-GL, as such a version would be similar to the recent DDGL approach Zhang *et al.* (2023) with only small differences.

The performance of v-GMM$^{\mathrm{d}}$-GL, v-MFA-GL, and MFA-GL is compared to all distribution-based methods benchmarked for the DDGL approach (cf. Zhang *et al.*, 2023), namely AGR (Liu *et al.*, 2010), EAGR (Wang *et al.*, 2016), f-FME (Qiu *et al.*, 2019), and DDGL (Zhang *et al.*, 2023) itself. Furthermore, we compare to MiMoLaP$^{\mathrm{LG}}$, MiMoLaP$^{\mathrm{H}}$ (Chi *et al.*, 2010) as the implementation of these methods is straight-forward, and we compare to r-FME (Qiu *et al.*, 2019) for which source code is available.

The above listed distribution-based GSSL algorithms are used for all comparisons on all benchmarks throughout this section. We evaluate efficiency and effectiveness of each algorithm using one fixed set of hyperparameter values used across all above benchmark datasets and settings. Details on how the hyperparameters of each algorithm were chosen are given in Appendix C.1.3. In Appendix C.2.2, we provide comparisons to still further approaches, including both further distribution-based GSSL methods and graph neural networks.

To infer the labels, the GSSL algorithms are applied in the experiments as follows: In the distribution learning stage, we train the distribution model (k-means or GMM) until a convergence criterion is reached (see Appendix C.1.3). We then fix the distribution model, and execute the graph learning and labeling stage using a number of labeled data points N_l within the range specified in Tab. 4.2. In order to obtain statistics on the performance, we learn 40 different distribution models using 40 different initializations. For each learned distribution models, we execute the graph learning and labeling stage 20 times per N_l, each time using a different random realization of the N_l initially labeled data points. Thus, for each dataset and each number of labeled data points N_l, we infer the labels of the N_u unlabeled data points $40 \cdot 20 = 800$ times.

To assess the effectiveness of a given algorithm for a given number of labeled data points N_l, we compute its error rate, i.e., the rate with which the inferred labels of the N_u unlabeled data points differ from the ground-truth labels. For each algorithm and dataset, the 800 runs thus result in 800 error rates for each N_l. We use these error rates to provide the median and the standard error of the median via bootstrap resampling (see Sec. 4.3.2 and Sec. 4.3.3). First, we will investigate the algorithms' runtimes, however.

4.3.1 Runtime Comparison

While previous work (e.g., Qiu *et al.*, 2019; Zhang *et al.*, 2023) focused on evaluating the algorithmic efficiency during the graph learning and labeling stage, we compare the runtime of the algorithms for both the distribution learning stage and the graph learning and labeling stage. The algorithms AGR, EAGR, f-FME, and r-FME use the k-means algorithm which learns cluster centers that can in our context be considered as components of a constrained GMM (cf. Lücke and Forster, 2019). Due to k-means being very efficient, AGR, EAGR, f-FME, and r-FME can be expected to achieve better runtimes in the distribution learning stage compared to DDGL, MiMoLaP$^{\mathrm{LG}}$, and MiMoLaP$^{\mathrm{H}}$, which rely on GMMs with diagonal covariance matrices. Still more so, all above stated algorithms can be expected to achieve better runtimes compared to MFA-GL, which is based on the most flexibly parameterized GMMs that we here consider.

The v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL algorithms exhibit reduced algorithmic complexity due to the applied variational techniques. Previous work in the context of clustering has shown that applying variational EM to GMMs with diagonal covariances (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022) reduces the runtime complexity of those models from scaling linearly with the product NC to a sublinear scaling; and the same has recently been shown for MFA models (Salwig *et al.*, 2025). The question arises, however, if the reduced complexity of the variational algorithms do also result in significant reductions of runtimes when they are used within a GSSL framework, and if they can outperform algorithms based on k-means.

For the graph learning and labeling stage, Zhang *et al.* (2023) reported that DDGL outperforms several other GSSL methods, including AGR, EAGR and f-FME. Although it can be expected that the distribution learning stage will take significantly longer than the graph learning and labeling stage for the evaluated algorithms, it remains to be assessed whether the applied variational techniques in v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL also achieve speed-ups in the graph learning and labeling stage.

To assess the efficiency of all algorithms, we measured the wall-clock runtimes of the algorithms for both distribution learning as well as for the graph learning and labeling stage (excluding data preprocessing). Wall-clock runtimes encompass all computational processes, memory management, and auxiliary routines, rendering any improvements in this metric particularly significant for practical applications. However, it is important to note that the exact runtime values are inherently influenced by implementation specifics and the hardware employed. Nevertheless, observed trends, such as a 'Method A' requiring significantly more time than a 'Method B', can be expected to generalize.

The experiments followed the experimental scheme described above. Each algorithm uses the same default values for its hyperparameters for all seven benchmarks. We used for all benchmarks and for all algorithms the same number of components, namely $C = 1000$. The default values of the other, partly specific, hyperparameters for each algorithm are provided in Appendix C.1.3. We measured 800 times the corresponding runtime of one algorithm run, including both distribution learning stage and graph learning and labeling stage, per N_l. We used then all measured runtimes to provide the median and the standard error of the median via bootstrap resampling.

The runtimes for all algorithms and benchmarks are given in Fig. 4.4. While accuracies will be systematically compared further below, the following observations can be made by considering the runtimes alone:

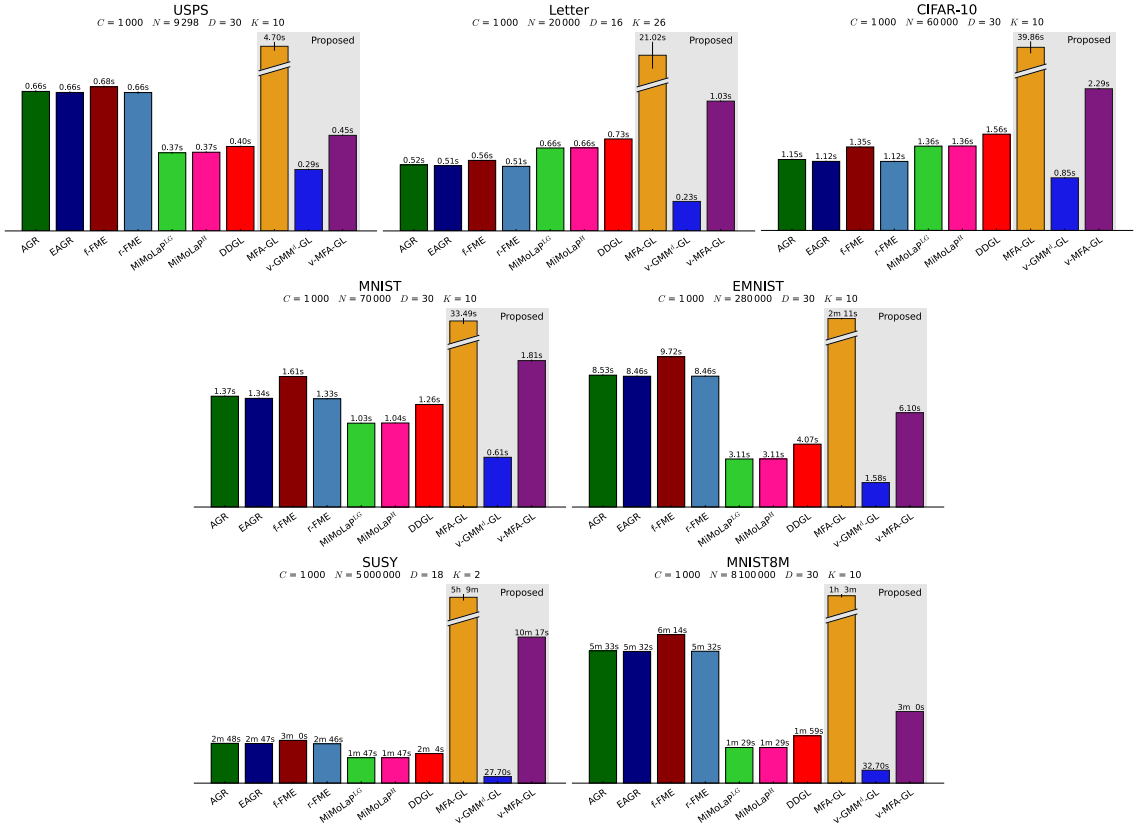

Fig 4.4: Runtimes of ten different GSSL methods on seven standard benchmarks. Each of the seven subfigures corresponds to experiments on one benchmark dataset. The methods from the literature include AGR, EAGR, f-FME, r-FME, MiMoLaP$^{\mathrm{LG}}$, MiMoLaP$^{\mathrm{H}}$, and DDGL, while the newly proposed methods are MFA-GL, v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL (indicated by a gray background). We measured the runtime of a single algorithm run 800 times per N_l. Each runtime in the figure represents the median runtime calculated over all these measurement points. The standard errors (error bars) of the median were calculated using bootstrap resampling.

AGR, EAGR, f-FME, and r-FME consistently exhibit comparable runtimes, which can be attributed to their reliance on the same distribution model (i.e., k-means). The approach that consistently showed the shortest runtimes across all benchmarks was v-GMM$^{\mathrm{d}}$-GL. MFA-GL was consistently the method with the (by far) longest runtimes. The algorithm's long runtime is particularly evident for the SUSY benchmark, where it exceeds 5 hours, compared to less than 11 minutes for all other methods, and less than 30 seconds for v-GMM$^{\mathrm{d}}$-GL. In contrast, the v-MFA-GL algorithm, which uses the same flexible distribution models as MFA-GL, shows much shorter runtimes which are similar to those of the other algorithms.

For the USPS, MNIST, EMNIST, SUSY, and MNIST8M datasets, it is notable that MiMoLaP$^{\mathrm{H}}$, MiMoLaP$^{\mathrm{LG}}$, and DDGL, which rely on GMMs, achieve shorter runtimes compared to AGR, EAGR, f-FME, and r-FME, which use k-means. This can be attributed to the differing number of iterations required for convergence: although k-means is faster per iteration, it can be observed to require significantly more iterations to converge on these datasets.

In Appendix C.3.1, we provide further details on the runtimes, including a detailed breakdown of runtimes in the distribution learning stage and graph learning and labeling stage for all algorithms.

We note that Zhang *et al.* (2023) suggested that the transformation matrix $\boldsymbol{Z}$ can be intuitively considered sparse, which can accelerate the graph learning and labeling stage of DDGL as well as MFA-GL. Such a version of DDGL and MFA-GL is analyzed in Appendix C.3.1. However, the most time-consuming part of DDGL and MFA-GL remains the distribution learning stage (training of the GMMs). Consequently, we remark that the accelerated graph learning and labeling stage has a negligible effect on the overall runtime of DDGL and MFA-GL.

4.3.2 Accuracy Comparison

While runtime efficiency was the primary focus of the previous subsection, we now systematically investigate the quality of all algorithms. We compare the proposed methods, including v-GMM$^{\text{d}}$-GL and v-MFA-GL (as well as MFA-GL) against the same set of algorithms evaluated in Sec. 4.3.1. The comparison is based on classification error rates across the benchmarks and across different numbers of labeled data points per dataset (cf. Tab. 4.2). We used the same datasets, and we applied all algorithms following the same experimental procedure and using the same default hyperparameters as used in Sec. 4.3.1.

Figure 4.5 shows the classification error rates for all algorithms and datasets, depending on the number of labeled data points N_l. In addition, we provide these error rates in tabular form in Appendix C.3.2. Dashed lines in Fig. 4.5 indicate previous methods from the literature, while solid lines represent the newly proposed methods. For r-FME, we observed in some settings that error rates increased with increasing numbers of labeled data points. The observation can be attributed to the sensitivity of r-FME hyperparameters to the number of labels. In the original work (Qiu *et al.*, 2019) the sensitivity was addressed by choosing different hyperparameter values for different numbers of labeled data points. For a consistent comparison, we used for all algorithms the same hyperparameters for all numbers N_l (which results in observed increase for r-FME).

For the here introduced v-GMM$^{\text{d}}$-GL algorithm, it can be observed that it consistently achieves lower error rates compared to the previous methods. The only exception occurs on the SUSY dataset in the case when 60 or more labeled data points are used. In that case, the error rates of DDGL are slightly lower than for v-GMM$^{\text{d}}$-GL (but the rates are almost identical). In six of the seven benchmarks, also v-MFA-GL achieves error rates that are lower than those of previous methods. On USPS, Letter, MNIST, EMNIST, and MNIST8M, either v-MFA-GL or MFA-GL shows the smallest errors of all algorithms, improving the error rate of v-GMM$^{\text{d}}$-GL still further (which already improved on the previous state-of-the-art). For the Letter, MNIST and EMNIST datasets, the improvements are very substantial, especially considering such well-known benchmarks. Furthermore, we remark that for USPS, MNIST, EMNIST, and MNIST8M, the error rates of v-MFA-GL remain almost constant even when the number of labels N_l becomes very small. On the other hand, on SUSY, v-MFA-GL (as well as MFA-GL) exhibits larger error rates than all other methods.

The largest here used dataset, MNIST8M, is an extension of standard MNIST obtained through data augmentation (see Appendix C.1.1). For all algorithms, we observe larger error rates on MNIST8M than for MNIST. We remark in this context that data augmentation

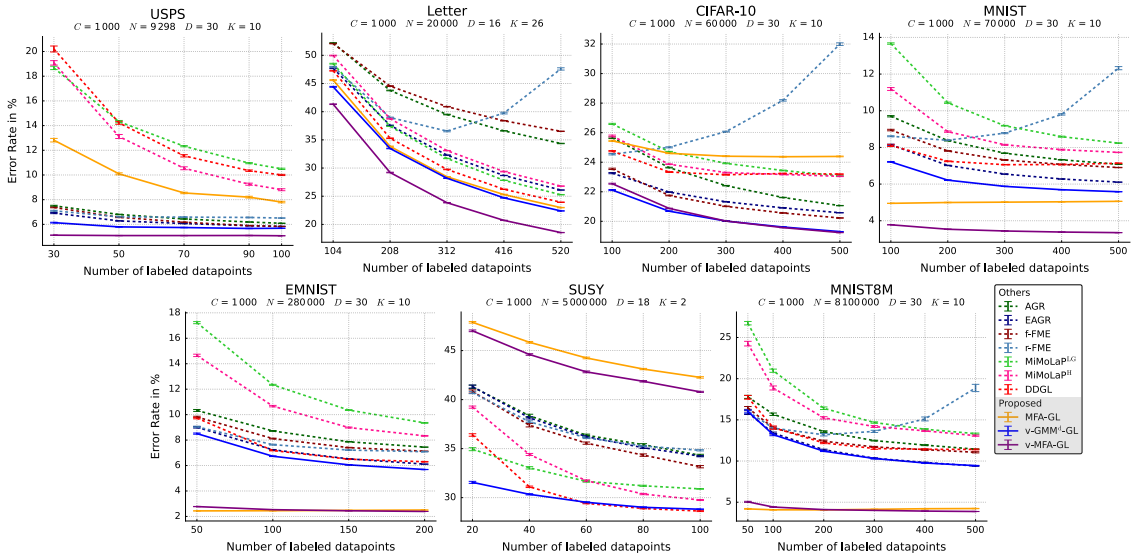

Fig 4.5: Error rates in % for varying numbers of labeled data points of ten different GSSL methods. Each of the seven subfigures corresponds to experiments on one benchmark dataset. The methods from the literature include AGR, EAGR, f-FME, r-FME, MiMoLaP$^{\mathrm{H}}$, MiMoLaP$^{\mathrm{LG}}$, and DDGL (dashed lines), while the newly proposed methods are MFA-GL, v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL (solid lines). Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling. The legend applies to all seven subfigures.

makes the here addressed SSL task more complex (the data distribution becomes more complex, and also augmented data has to be labeled). While data augmentation is typically expected to improve performance in other settings, small increases in error rates should therefore not be surprising here. However, if the number of components C is increased to match the bigger and more intricate data of MNIST8M, the error rates of v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL decreases again (for details see Appendix C.2.4, where we went up to $C = 10\,000$).

In summary of all numerical experiments, we can state that the error rate of v-GMM$^{\mathrm{d}}$-GL shows in all experiments lower (or nearly identical) error rates compared to all previous approaches. By considering the runtime measurements of Sec. 4.3.1, v-GMM$^{\mathrm{d}}$-GL thus improves performance and, at the same time, substantially improves the runtime compared to the previous state-of-the-art in GSSL. For v-MFA-GL, we can state that in almost all settings error rates are further improved, and for some datasets by large margins.

4.3.3 The Limit of a Single Label per Class

In Sec. 4.3.2, we showed that the proposed algorithms, v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL, outperformed the previous algorithms in terms of classification error rate across nearly all datasets and settings. Although the number of labeled data points was relatively low, which already posed a significant challenge, each setting contained at least three data samples per label class, with most settings including more than three samples.

In this section, we further challenge the algorithms by evaluating them in an extreme scenario where only a single labeled data point is available per class. In such a case, the algorithms' performance is highly dependent on the quality of the provided samples. If a

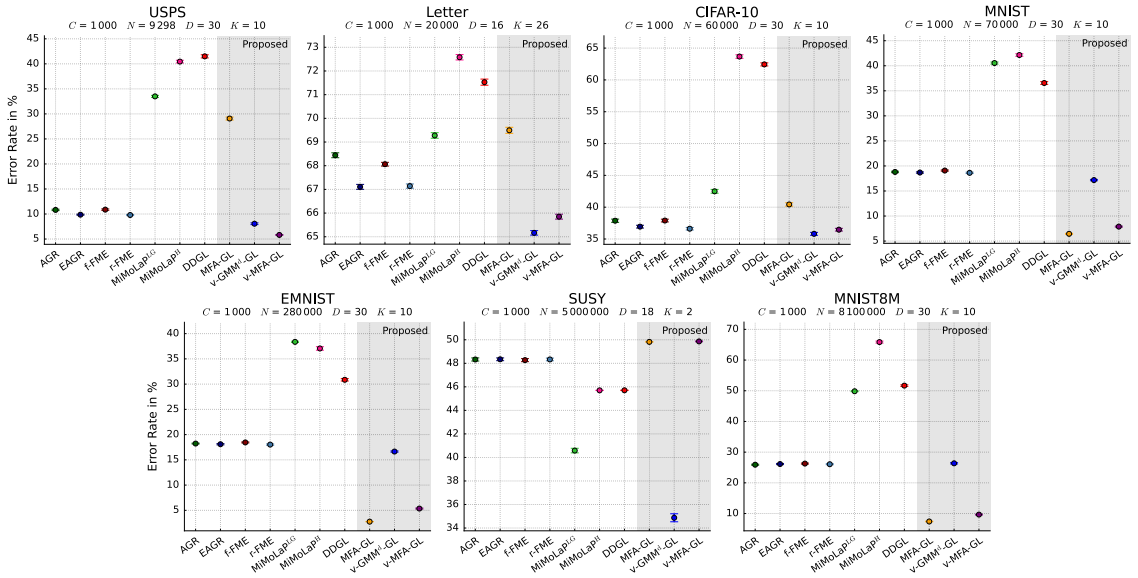

Fig 4.6: Error rates in % for ten different GSSL methods, using only a single labeled data point per class. Each of the seven subfigures corresponds to experiments on one benchmark dataset. The methods from the literature include AGR, EAGR, f-FME, r-FME, DDGL, MiMoLaP$^{\mathrm{H}}$, and MiMoLaP$^{\mathrm{LG}}$, while the newly proposed methods are MFA-GL, v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL (indicated by a gray background). Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling.

sample does not adequately represent its class, the algorithm may struggle to accurately predict the labels for that class.

To address this challenge, it is essential to effectively leverage information from the unlabeled data points while dealing with the severely limited labeled data. To evaluate the efficacy of each method in this scenario, we repeated the experiments from Sec. 4.3.2, this time only providing a single labeled sample per class. Figure 4.6 presents the classification error rates for the different algorithms in this setting.

The error rates of all algorithms increased compared to the smallest number of labeled data points in Fig. 4.5. The v-GMM$^{\mathrm{d}}$-GL algorithm achieved lower error rates than all previous methods in all datasets except MNIST8M. Furthermore, v-GMM$^{\mathrm{d}}$-GL attained the lowest overall error rates on the Letter, CIFAR-10, and SUSY datasets. For the remaining datasets, USPS, MNIST, EMNIST, and MNIST8M, either v-MFA-GL or MFA-GL yielded the lowest error rates. The AGR, EAGR, f-FME, and r-FME algorithms, all of which utilize k-means, showed error rates that were similar to each other across all benchmark datasets.

4.4 Conclusion

We have investigated novel approaches to graph-based semi-supervised learning (GSSL). Our core methodological contribution is the integration of variational optimization techniques. By considering the derivation of SSL methods based on variational posteriors (Sec. 4.2.3), it can be observed that the here applied truncated posteriors, which are themselves motivated by properties of natural data (Hughes and Sudderth, 2016; Lücke and Eggert, 2010; Shelton *et al.*, 2017), seamlessly integrate into the GSSL framework: They address the main

bottleneck of distribution learning, and truncated posteriors naturally provide sparse graphs that are well suited for effective and efficient label propagation. Sparse graphs (i.e., sparse graph Laplacians) have been used before (Liu *et al.*, 2010; Qiu *et al.*, 2019; Wang *et al.*, 2016) but the sparsity was previously introduced *ad hoc*. It could thus be argued that the sparse graphs here applied are naturally consistent with the learned data representations.

For the label propagation stage, we empirically observed that the information about *which* components are connected to each other (given by the positions of the non-zero entries in the transformation matrix $\boldsymbol{Z}$) is more important than the strengths of these connections (i.e., the edge weights which are inferred from the values of $\boldsymbol{Z}$, cf. Eq. (4.5)). Even if all values of the non-zero entries of $\boldsymbol{Z}$ were normalized to the same value, performance remained high (see Appendix C.2.4). However, performance decreases if the graph gets overly sparse, i.e., if edges are suppressed. The here suggested use of hot posteriors (defined in Sec. 4.2.5) prevents overly sparse graphs by avoiding small non-zero values in the transformation matrix $\boldsymbol{Z}$. Consequently, the use of hot posteriors is observed to be important for performance (see Appendix C.2.3). And we note that such hot posteriors are directly tied to the used truncated posteriors, as only non-zero values are affected. Sparse graphs are, however, not only important for effective but also for efficient label propagation. They thus, e.g., facilitate the computation of the soft class matrix $\boldsymbol{A}$ using conjugate gradients (see Sec. 4.2.3).

Together with the performance enhancements induced by sparse graphs, an important benefit of truncated posteriors is, of course, the majorly enhanced efficiency during the distribution learning stage. For the v-GMM$^{\text{d}}$-GL algorithm the above discussed contributions do, consequently, not only improve the SSL performance compared to the state-of-the-art (see Fig. 4.5). At the same time, v-GMM$^{\text{d}}$-GL is much more efficient and substantially decreases runtimes (see Fig. 4.4). For instance, on the large-scale GSSL benchmarks EMNIST, SUSY and MNIST8M, v-GMM$^{\text{d}}$-GL is about twice or even multiple times as fast as previous approaches.

Importantly, the improvements in runtimes do not only allow for accelerating GSSL with previously used distribution models. They do also allow for using enhanced distribution models. Scalable distribution-based GSSL methods either used k-means, or they were using GMMs with isotropic or diagonal covariance matrices as distribution models. Consequently, no within-components correlations could be modeled. If data lies close to low dimensional manifolds, a distribution model able to match low-dimensional correlation structure can, consequently, be expected to improve GSSL (cf. Fig. 4.1 for an illustration). With the integration of mixtures of factor analyzers (MFAs) into GSSL, we have here investigated the first GSSL method that incorporate the manifold hypothesis directly into the distribution model (but compare Qiu *et al.*, 2019, for manifold embedding at a later stage). Initially, the price to pay is a much longer execution time of the corresponding MFA-GL algorithm (see Fig. 4.4). However, by using truncated posteriors in conjunction with the further improvements in label propagation, we obtain an efficient and very effective novel GSSL algorithm in the form of v-MFA-GL. For five of six previously used standard benchmarks (cf. Zhang *et al.*, 2023), v-MFA-GL establishes a new state-of-the-art in GSSL performance (i.e., new lowest error rates). On the SUSY benchmark, however, only the v-GMM$^{\text{d}}$-GL algorithm sets a new state-of-the-art performance (for few labels), while v-MFA-GL (and MFA-GL) even perform significantly worse than all other approaches. The most likely reason is that assumptions related to the submanifold hypothesis are not fulfilled by the SUSY dataset.

On the other hand, especially for MNIST and EMNIST (but less pronounced also for USPS and Letter) we observed a substantial performance improvement of v-MFA-GL compared to the state-of-the-art. On MNIST with 100 labels, v-MFA-GL halves the previous state-of-the-art error rate from about 8% to about 4%. On EMNIST, v-MFA-GL maintains an error rate below 3% even with only 50 labels, while the previous state-of-the-art exceeds 8% for 50 labels and still exceeds 6% for 200 labels (also compare Zhang *et al.*, 2023). Such big steps in performance improvements are notable for established benchmark datasets, and it can here be attributed to the manifold hypothesis which seems to apply well, e.g., for MNIST or EMNIST. Remarkably, the modeling flexibility of MFAs alone is sometimes not sufficient for state-of-the-art performance (see, e.g., MFA-GL performance on USPS). However, the flexibility of MFAs in combination with improvements induced by hot truncated posteriors seems to reliably result in the high performance of v-MFA-GL (given the manifold hypothesis is fulfilled). Performance increases by enhanced distribution models are confirmed for the large-scale variant of MNIST. Concretely, on the MNIST8M benchmark the error rate of v-MFA-GL remains at a value of 5% or below even if the number of labels decreases to 50 labels. In comparison, previous algorithms were observed to remain at an error rate of above 15% in this case (see Fig. 4.5).

As an outlook we remark that the scaling of v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL to much larger distribution models and graphs results in little additional effort. The sublinear scaling of v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL thus makes runtimes just two or three times longer if the number of components is scaled from $C = 1000$ to $C = 10\,000$. Future work could further exploit such sublinear scaling, e.g., by learning increasingly refined data and graph representations on augmented data. The example of MNIST8M may provide first evidence in favor of such future strategies: with increasingly large distribution models and graphs, we observe a continuous further improvement in labeling accuracy (cf. Appendix C.2.4).

Chapter 5

Variational Zero-Shot Denoising using Mixture of Factor Analyzers

In this chapter, we investigate the application of the variational optimization techniques introduced in Ch. 3 for optimizing probabilistic generative models in the task of blind zero-shot denoising (cf. Ch. 2). Concretely, we apply the v-MFA algorithm to the denoising of macroscopic natural images.

This chapter presents research that has not been published, so far. The derivation of the probabilistic data estimator for v-MFA and v-GMM$^{\mathrm{d}}$, as well as the denoising experiments involving these algorithms, were done without contributions of others. There were also no other authors for the text of this chapter and its associated tables.

5.1 Introduction

Denoising is one of the core tasks in signal and image processing, and it is widely applied in various domains, such as computer vision, medicine and audio processing. It refers to the process of removing noise from data including, e.g., images or audio recordings, to recover the underlying, uncorrupted information. However, the task becomes particularly challenging in scenarios where no clean data is available, when only a limited number of noisy samples (potentially as few as a single sample) are present or neither the noise type nor its statistical properties or strength are known.

In such settings, blind zero-shot denoising algorithms are particularly relevant, as they are capable of operating directly on a single noisy sample without requiring external training data or prior knowledge about the noise characteristics. In Ch. 2, we have investigated a range of blind zero-shot algorithms applied to microscopy imaging data. Concretely, we evaluated elementary and advanced filter-based approaches (e.g., Azzari and Foi, 2016b; Dabov *et al.*, 2007; Makitalo and Foi, 2011), deep neural networks (DNNs) with objectives adapted for zero-shot applicability (e.g., Krull *et al.*, 2019a; Lequyer *et al.*, 2022b; Quan *et al.*, 2020a) and probabilistic generative models (e.g., Drefs *et al.*, 2022; Prakash *et al.*, 2021b, and Poisson mixture models), which are inherently well-suited for scenarios involving only noisy data.

The results in Figs. 2.2 to 2.4 suggest that the suitability of a blind zero-shot denoising algorithm for a given microscopy imaging dataset is influenced by several factors, for example,

the degree to which the algorithm's assumptions align with the statistical properties of the data. No single algorithm or class of algorithms consistently proved to be the most suitable across all scenarios. However, the findings also indicate that blind zero-shot methods, particularly the DNN-based methods (Krull *et al.*, 2019a; Quan *et al.*, 2020a) or, e.g., the generative model based ES3C approach (Drefs *et al.*, 2022), can exhibit long runtimes or high computational costs (compare Tab. A.2). This limits their practicality for real-time or time-sensitive applications such as live-cell experiments in microscopy, where the microscope must respond to dynamic changes (Lequyer *et al.*, 2022b), or PET imaging in medicine, where timely processing is critical for prompt clinical decision-making (Hellwig *et al.*, 2023).

In this chapter, we therefore investigate the application of the novel variational optimization techniques introduced in Ch. 3 for optimizing probabilistic generative models in the task of denoising. Concretely, we apply the v-MFA algorithm which integrates the variational EM algorithm with mixtures of factor analyzers (MFAs) to the denoising of macroscopic natural images under blind zero-shot conditions. To adapt v-MFA for denoising, we embed it within the patch-based restoration framework that is also employed by both ES3C and the PMM-based approach in Ch. 2 (also compare Appendix A.1). This restoration framework relies on a probabilistic data estimator, which leverages learned data representations to estimate the underlying clean image from noisy observations (for more details, see Sec. 5.2.1). Thereby, the employed variational optimization techniques not only enable efficient training of MFAs but also facilitate an efficient approximation of the data estimator, resulting in a highly efficient denoising pipeline.

Although the MFA model is neither explicitly designed nor tuned for denoising, it can play a valuable role in the context of patch-based denoising as it enables the representation of intra-component correlations along low-dimensional subspaces. Image patches are commonly assumed to lie near such low-dimensional subspaces or *manifolds* (e.g., Lee *et al.*, 2003; Peyré, 2009), and it has been shown that patch models that yield high likelihoods to patches drawn from natural images achieve a high performance in patch and image restoration tasks (e.g., Drefs *et al.*, 2022; Zoran and Weiss, 2011). Consequently, models capable of capturing these low-dimensional structures, such as MFAs, can be expected to improve denoising performance.

The results presented in this chapter constitute the first evaluation of variationally trained MFAs and GMMs (as described in Ch. 3) on the task of denoising. As an initial benchmark, we adopted the evaluation protocol from Lequyer *et al.* (2022b), which evaluates various zero-shot denoising algorithms based on deep neural networks (DNNs), focusing on both denoising performance and computational efficiency. It is important to note, however, that this benchmark exclusively considers DNN-based methods and does not include non-DNN approaches, such as generative models or filtering techniques. Consequently, the evaluation presented here is not intended to provide a comprehensive comparison across different classes of denoising algorithms, as in Ch. 2. Instead, it serves as an initial reference point for positioning v-MFA relative to current state-of-the-art DNN-based methods. This comparison is also particularly relevant given the dominant role that DNN-based approaches currently play in advancing the fields of machine learning and artificial intelligence.

5.2 Methods

In the following section, we describe how a single noisy image can be reconstructed using the MFA model and the v-MFA algorithm.

5.2.1 Probabilistic Data Estimation with MFAs

The generative model of mixtures of factor analyzers (MFAs; Ghahramani and Hinton, 1996; Richardson and Weiss, 2018) is given by:

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{z} \,|\, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \boldsymbol{I}), \qquad p(\boldsymbol{x} \,|\, \boldsymbol{z}, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c, \boldsymbol{D}_c), \quad (5.1)$$

where $\pi_c \in [0, 1]$ are the mixing proportions, $\boldsymbol{\mu}_c \in \mathbb{R}^D$ are the means, $\boldsymbol{\Lambda}_c \in \mathbb{R}^{D \times H}$ is the factor loading matrix and $\boldsymbol{D}_c \in \mathbb{R}^{D \times D}$ denotes a diagonal covariance matrix.

To perform image denoising using MFAs, we extract overlapping patches from the input image, which serve as the training data. Parameter estimation is then performed using the variational EM algorithm, as detailed in Ch. 3 and Alg. 2.

Once the model parameters have been optimized, the learned MFA representations are employed to estimate the underlying non-noisy image patches. This reconstruction uses a probabilistic data estimator based on the posterior predictive distribution $p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})$ (Drefs *et al.*, 2022). For the MFA model, the posterior predictive distribution is given by:

$$p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) = \sum_c \int p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{z}, \boldsymbol{\Theta}) \; p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \; d\boldsymbol{z}. \quad (5.2)$$

The denoised pixel values of a given image patch $\boldsymbol{x}_n$ are then obtained by taking the expectation w.r.t. this posterior:

$$\begin{aligned} \mathbb{E}_{p(\boldsymbol{x}_n^{\text{est}} \,|\, \boldsymbol{x}_n, \boldsymbol{\Theta})}\big[\boldsymbol{x}_n^{\text{est}}\big] &= \mathbb{E}_{p(c \,|\, \boldsymbol{x}_n, \boldsymbol{\Theta})}\big[\boldsymbol{\Lambda}_c \boldsymbol{V}_c (\boldsymbol{x} - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c\big] \\ \text{with} \quad \boldsymbol{V}_c &= (\boldsymbol{I} + \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} \boldsymbol{\Lambda}_c)^{-1} \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} \end{aligned} \quad (5.3)$$

The detailed derivation of the probabilistic data estimator is provided in Appendix D.1. The expectation can then be efficiently approximated by replacing the exact posteriors with the truncated posteriors defined in Eq. (3.4), as follows:

$$\mathbb{E}_{p(\boldsymbol{x}_n^{\text{est}} \,|\, \boldsymbol{x}_n, \boldsymbol{\Theta})}\big[\boldsymbol{x}_n^{\text{est}}\big] \approx \mathbb{E}_{q_n(c; \boldsymbol{\mathcal{K}}, \boldsymbol{\Theta})}\big[\boldsymbol{\Lambda}_c \boldsymbol{V}_c (\boldsymbol{x}_n - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c\big] \quad (5.4)$$

After estimating a clean version of each extracted patch, the image is reconstructed by merging these patches. Because overlapping patches are used, each pixel in the image has multiple estimates. Consequently, the final pixel value is determined by computing the median of all corresponding estimates for that pixel (see also Fig. A.1 for an illustration of the denoising pipeline).

5.3 Results

To evaluate the denoising performance and computational efficiency of v-MFA, we follow the benchmarking protocol of Lequyer *et al.* (2022b) using the two macroscopic natural image datasets BSD68 (Martin *et al.*, 2001) and Set12. BSD68 comprises 68 grayscale macroscopic natural images with a resolution of 481×321 pixels, while Set12 contains 12

grayscale macroscopic natural images with varying resolutions ranging from 256×256 to 512×512 pixels. Noisy versions of the images are generated by adding synthetic Gaussian noise with the standard deviations σ listed in Tab. 5.1.

The algorithms evaluated in this benchmark consist of various zero-shot denoising algorithms based on DNNs. These algorithms include Noise2Self (N2S; Batson and Royer, 2019), Noise2Void (N2V; Krull *et al.*, 2019a), Self2Self (S2S; Quan *et al.*, 2020a), Neighbor2Neighbor (Ne2Ne; Huang *et al.*, 2021), deep image prior (DIP; Ulyanov *et al.*, 2018) and Noise2Fast (N2F; Lequyer *et al.*, 2022b). Among these, N2V, S2S and N2F have also been assessed in Ch. 2.

To assess the effect of modeling correlations through MFAs, we additionally include in this benchmark a Gaussian mixture model (GMM) with diagonal covariance matrices, which cannot capture inter-component correlations. We integrate the GMM into the same patch-based restoration framework that is employed by v-MFA. We will refer to this method as v-GMM$^{\mathrm{d}}$ in the following. Details on the denoising procedure using GMMs, including the derivation of the probabilistic data estimator, are provided in Appendix D.2.

The zero-shot denoising algorithms including the DNN-based approaches as well as v-MFA and v-GMM$^{\mathrm{d}}$ are independently applied to each noisy image of the sets, 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. These quantitative metrics are then averaged across all images in the dataset. The results for the DNN-based methods are taken from Lequyer *et al.* (2022b), who conducted the evaluations on an NVIDIA RTX 5000 mobile GPU. For additional details, including the hyperparameter settings, we refer to Lequyer *et al.* (2022b).

To apply v-MFA and v-GMM$^{\mathrm{d}}$ to the benchmark datasets, we employed the default hyperparameters as previously used in Chs. 3 and 4, without any dataset-specific or noise-level fine-tuning. Concretely, we set the number of mixture components to $C = 1000$ and the dimensionality of the hyperplane to $H = 5$ (only for v-MFA). The variational hyperparameters are chosen as $C' = 3$ and $G = 15$. The patch size was fixed at $D = 12 \times 12$ across the datasets, a size shown to perform well for macroscopic natural image denoising in previous work (Drefs *et al.*, 2022). We conducted the experiments using v-MFA and v-GMM$^{\mathrm{d}}$ on an AMD Genoa EPYC 9554 CPU. To more closely simulate the moderate (albeit still substantial) computational resources typical of an average end user, we ran four instances of v-MFA concurrently. Each instance utilized 16 out of the 64 available CPU cores, resulting in four independent parallel executions. The results are presented in Tab. 5.1.

The results show that v-MFA consistently ranks among the top three evaluated methods in terms of denoising performance. The best-performing method is always S2S, which surpasses v-MFA with PSNR differences ranging from 0.21 to 1.59 across the evaluated datasets and noise levels. Notably, this performance gap narrows as the noise level increases. Among the remaining methods (excluding S2S), v-MFA generally achieves superior performance, except at noise level $\sigma = 15$ on BSD68 and Set12, where N2F attains higher PSNR values than v-MFA.

In terms of runtime, v-GMM$^{\mathrm{d}}$ is consistently the fastest method, followed by v-MFA. Compared to the fastest DNN-based method (N2F), v-MFA achieves a speed-up of more than

Tab 5.1: PSNR, SSIM and average runtime per image (in seconds) for eight blind zero-shot denoising algorithms evaluated on the Set12 and BSD68 datasets for different noise levels σ. Results for N2S, N2V, DIP, Ne2Ne, S2S and N2F are taken from Lequyer *et al.* (2022b). Each value (PSNR, SSIM and runtime) represents the mean over all images in the corresponding dataset. For v-GMM$^{\mathrm{d}}$ and v-MFA, the standard error of the mean (SEM) is also reported. Error metrics for the remaining methods were not provided in Lequyer *et al.* (2022b). Note that the SSIM values are scaled by a factor of 10.

Dataset	σ		N2S	N2V	DIP	Ne2Ne	S2S	N2F	v-GMM$^{\mathrm{d}}$	v-MFA
Set12	15	PSNR	30.69	30.04	28.51	27.97	**32.17**	31.10	26.91 ± 0.60	30.58 ± 0.50
		SSIM	8.71	8.47	8.04	7.87	**8.89**	8.71	7.90 ± 0.15	8.67 ± 0.10
		Time per Image in s	2161.00	2682.00	70.00	198.00	9484.00	22.00	$\mathbf{0.88 \pm 0.15}$	3.71 ± 0.76
	25	PSNR	28.35	28.41	26.47	26.23	**29.88**	29.05	26.62 ± 0.55	29.16 ± 0.41
		SSIM	7.76	7.80	7.07	6.87	**8.42**	8.22	7.78 ± 0.14	8.26 ± 0.10
		Time per Image in s	2161.00	2682.00	70.00	198.00	9484.00	18.00	$\mathbf{0.83 \pm 0.14}$	3.39 ± 0.67
	35	PSNR	26.59	27.29	24.25	25.09	**28.24**	27.57	26.16 ± 0.50	27.82 ± 0.36
		SSIM	7.27	7.71	5.96	6.32	**7.99**	7.81	7.59 ± 0.14	7.76 ± 0.10
		Time per Image in s	2161.00	2682.00	71.00	198.00	9484.00	19.00	$\mathbf{0.81 \pm 0.13}$	3.28 ± 0.61
	50	PSNR	25.04	25.70	21.19	23.43	**26.34**	25.82	25.32 ± 0.46	26.13 ± 0.34
		SSIM	6.87	7.20	4.39	5.32	**7.34**	7.23	7.25 ± 0.15	7.04 ± 0.11
		Time per Image in s	2161.00	2682.00	70.00	198.00	9484.00	21.00	$\mathbf{0.79 \pm 0.12}$	3.42 ± 0.63
BSD68	25	PSNR	27.50	26.66	25.74	25.68	**28.70**	28.12	25.38 ± 0.42	27.25 ± 0.40
		SSIM	7.73	7.31	6.85	6.92	**8.03**	7.89	6.71 ± 0.16	7.47 ± 0.11
		Time per Image in s	1619.00	2682.00	69.00	231.00	7962.00	29.00	$\mathbf{0.80 \pm 0.01}$	3.58 ± 0.05
	50	PSNR	24.53	24.50	21.29	23.59	**25.92**	25.23	24.50 ± 0.37	25.30 ± 0.32
		SSIM	6.46	6.05	4.42	5.45	**6.99**	6.70	6.31 ± 0.15	6.52 ± 0.11
		Time per Image in s	1619.00	2682.00	69.00	231.00	7962.00	26.00	$\mathbf{0.81 \pm 0.01}$	3.80 ± 0.06

5×. Remarkably, when compared to S2S (the only method that consistently outperforms v-MFA in terms of denoising performance), v-MFA is over 2000× faster.

However, these runtime comparisons should be interpreted with caution for several reasons: First, the DNN-based methods were executed on different hardware than v-MFA and v-GMM$^{\mathrm{d}}$, which were both run on a CPU. Second, v-MFA and v-GMM$^{\mathrm{d}}$ do not match the denoising performance of, e.g., S2S, and it remains unclear how much faster S2S would be if it were only required to reach the (lower) performance level attained by v-MFA.

Notably, Lequyer *et al.* (2022b) compared the accuracy of each method when limiting the maximum number of iterations so that each algorithm runs only as long as N2F requires to fully denoise the images. We adapted this comparison in Tab. D.1 and added the results of v-MFA and v-GMM$^{\mathrm{d}}$ (which are substantially faster than N2F). Under these conditions, v-MFA and N2F achieve the best performances in terms of PSNR and SSIM values, depending on the dataset and noise level.

5.4 Discussion and Conclusion

In this chapter, we investigated the application of the v-MFA algorithm to macroscopic natural image denoising under zero-shot conditions. We compared v-MFA to a range of zero-shot denoising methods based on DNNs.

The results show that v-MFA is highly competitive in terms of denoising performance, as measured by PSNR and SSIM. This is particularly notable given that MFAs optimized by the v-MFA algorithm are neither specifically designed nor tuned for the task of image denoising. Remarkably, v-MFA consistently outperformed several DNN-based zero-shot

methods (e.g., N2V, N2S) across nearly all evaluated settings, despite these methods being explicitly developed and optimized for this task. Moreover, DNN-based approaches typically employ substantially larger patch sizes, thereby exploiting more contextual information (e.g., 64×64 in N2V, Krull *et al.* 2019a), which makes the performance of v-MFA even more remarkable given its relatively limited local context (with a patch size of 12×12).

The results further indicate that many DNN-based algorithms require a considerable amount of time to denoise a single image. In contrast, v-MFA and v-GMM$^{\mathrm{d}}$ achieve substantially lower runtimes on the order of only a few seconds. Although runtimes can vary depending on the hardware used, this efficiency suggests that these methods are particularly suitable for applications requiring fast processing, such as live-cell or PET image denoising. An exception among the DNN-based methods is the N2F algorithm, which was explicitly designed for fast convergence and demonstrated substantially shorter training times compared to other DNN-based approaches (but N2F remains considerably slower than v-MFA and v-GMM$^{\mathrm{d}}$).

A comparison between v-MFA and v-GMM$^{\mathrm{d}}$ highlights the advantage of modeling correlations along lower-dimensional subspaces via the MFA structure in the task of patch-based image denoising: v-MFA consistently outperforms v-GMM$^{\mathrm{d}}$ in terms of PSNR and SSIM values across nearly all evaluated settings – and, in several cases, by a substantial margin.

Given the strong denoising performance of MFAs and the fast optimization times of the v-MFA algorithm, it is worthwhile to further explore its potential in image denoising tasks. Future work could include a broader evaluation of zero-shot denoising algorithms, extending beyond DNN-based methods to also encompass, e.g., filter-based approaches and other generative models. Moreover, due to the sublinear scaling behavior of the v-MFA algorithm, its application could also be investigated in non-zero-shot settings, where efficient training on very large image corpora may enable the model to exploit additional statistical structure across images.

Chapter 6

Implementation and Parallelization of Sublinear Variational Learning Algorithms for Mixture Models

In this chapter, we present the technical design decisions and implementation challenges involved in developing a CPU-based, parallelized version of the variational optimization method introduced in Ch. 3.

This chapter was written in collaboration with Till Kahlke. A detailed description of each author's individual contributions is provided below.

Author Contributions. The numerical experiments reported in this chapter were done by Sebastian Salwig (SS) with input by Till Kahlke (TK). Figures and tables were created by SS with input from TK. The chapter was written by SS and TK. Both discussed the results and approved the final chapter version. For contributions to the software, see author contributions of Ch. 3.

6.1 Introduction

We here give details on the technical design choices and challenges associated with the CPU-based implementation and parallelization of sublinear learning algorithms including algorithms reported in the paper *"Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters"* from Salwig *et al.* (2025). The algorithms were derived from a variational optimization approach based on truncated variational distributions (e.g., Drefs *et al.*, 2022; Hirschberger *et al.*, 2022; Lücke, 2019), and they were applied by Salwig *et al.* (2025) to mixtures of factor analyzers (MFA; Ghahramani and Hinton, 1996; McLachlan *et al.*, 2003; Richardson and Weiss, 2018) as data models. This report gives details on the latest implementation, which was optimized for the parallel execution on many CPU-cores.

Throughout this document, 'variational EM' refers to *truncated variational Expectation Maximization*, which is a variational optimization approach for generative models with discrete latents (e.g., Drefs *et al.*, 2022; Hirschberger *et al.*, 2022; Mousavi *et al.*, 2023), and has its roots in earlier amortized forms of variational optimization (Lücke and Eggert,

2010; Sheikh *et al.*, 2014). While the detailed theoretical description of the variational EM algorithm can be found in the corresponding papers (detailed below), we here emphasize practical aspects and implementation details. The training principles and algorithm derivations for MFAs and GMMs are available in the papers Hirschberger *et al.* (2022) and Salwig *et al.* (2025). A formal introduction of truncated distributions within the framework of variational optimization is provided in Lücke (2019), and applications of truncated posteriors to models other than MFAs and GMMs are discussed, e.g., by Drefs *et al.* (2022, 2023) or Mousavi *et al.* (2023).

Design decisions for the here discussed implementations of MFA and GMM learning algorithms were primarily made with large-scale applications in mind (where practical runtime improvements become most relevant). The here described implementation is optimized for handling a large number of mixture components C and for training data containing a large number of data points N with high dimensionality D.

To ensure ease of use, we have developed a lightweight and user-friendly Python framework that builds upon a C++ implementation of the core functionalities of the variational EM algorithm. This framework currently supports the optimization of Gaussian mixture models (GMMs) with different types of covariance structures, including isotropic, diagonal and full variances, as well as mixtures of factor analyzers (MFAs). The variances can optionally be shared among all components. Moreover, the framework is designed for easy extension to incorporate other mixture models, which are planned for future work.

The technical report begins with a comparison of the implementation strategies between conventional and variational EM algorithms, followed by a detailed outline of the current implementation of the variational EM algorithm. Subsequently, we present an analysis of the practical performance of the variational EM algorithm through numerical experiments, which also shows bottlenecks in the current implementation.

6.2 Conventional vs. Variational EM Implementations

It has already been empirically demonstrated that the current implementation of the variational EM algorithm achieves faster wall-clock runtimes than an efficient implementation of the conventional EM algorithm, particularly as the number of components C increases (cf. Salwig *et al.*, 2025). However, for small values of C (e.g., $C \leq 10$), where both algorithms are expected to exhibit short runtimes, the conventional EM algorithm may outperform variational EM in terms of total runtime. This theoretically unexpected behavior can be attributed to practical hardware limitations, which we will discuss below.

The conventional EM algorithm is an iterative procedure consisting of two steps: the E-step, where the posterior probabilities of the latent variables are computed, and the M-step, where the model parameters are updated. Both steps involve computationally intensive operations, which can often be reformulated as linear algebra problems and efficiently implemented using vectorized matrix operations.

For instance, in the case of a Gaussian Mixture Model (GMM) with diagonal covariance matrices, a computationally demanding part of evaluating the posterior probabilities in the E-step is computing the squared Euclidean distance d^2 between each data point $\boldsymbol{x}_n$ and the mean $\boldsymbol{\mu}_c$ of each component c. This squared distance d^2 is given by:

$$d^2(\boldsymbol{x}_n, \boldsymbol{\mu}_c) = \|\boldsymbol{x}_n - \boldsymbol{\mu}_c\|^2 = \boldsymbol{x}_n^T \boldsymbol{x}_n - 2\boldsymbol{x}_n^T \boldsymbol{\mu}_c + \boldsymbol{\mu}_c^T \boldsymbol{\mu}_c \tag{6.1}$$

To efficiently compute the squared Euclidean distances for all data points and mixture components, we can store the dataset as an $N \times D$ matrix $\boldsymbol{X}$, where each row corresponds to a data point, and the component means as a $C \times D$ matrix $\boldsymbol{\mu}$, where each row corresponds to a component mean. The pairwise squared Euclidean distance $\boldsymbol{D}^2$ can then be efficiently computed using matrix operations:

$$\boldsymbol{D}^2(\boldsymbol{X}, \boldsymbol{\mu}) = \boldsymbol{X}^2 \mathbb{1}_{DC} - 2\boldsymbol{X}\boldsymbol{\mu}^T + \mathbb{1}_{ND}(\boldsymbol{\mu}^2)^T, \tag{6.2}$$

where $\mathbb{1}_{DC}$ and $\mathbb{1}_{ND}$ denote $D \times C$ and $N \times D$ matrices, respectively, where all entries are equal to one[6].

Due to the fundamental role of matrix multiplication in numerous numerical algorithms, a whole field of research is dedicated to optimize matrix multiplication algorithms for efficiency. Additionally, modern CPUs and GPUs are specifically optimized to accelerate matrix operations.

In practice, matrix multiplication is memory-bound, meaning the hardware spends more time loading data than performing computations. Naive implementations often fail to leverage CPU caches and memory hierarchies which results in poor performance. Optimized versions as used in libraries such as BLAS (Basic Linear Algebra Subprograms) and MKL (Math Kernel Library) improve cache utilization using techniques like loop blocking or tiling. These libraries provide highly optimized implementations of matrix multiplication, tailored to different hardware architectures, and are integrated into popular Python frameworks such as NumPy and PyTorch. Consequently, the conventional EM algorithm can efficiently be implemented in Python using such libraries.

However, the same implementations strategies do not apply for the variational EM algorithm. In the variational EM version, the posterior probabilities are replaced by truncated posterior probabilities. Rather than computing the posterior probabilities for all combinations of data points and components, the truncated version involves calculating the posterior probabilities only for a subset of components for each data point and setting the posteriors for the remaining components to zero. This truncated posterior probability is given by

$$q_n(c;\, \boldsymbol{\Theta}) := q(c; \boldsymbol{x}_n, \mathcal{K}^{(n)}, \boldsymbol{\Theta}) = \frac{p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta})}{\sum_{\tilde{c} \in \mathcal{K}^{(n)}} p(\tilde{c}, \boldsymbol{x}_n \mid \boldsymbol{\Theta})}\, \delta(c \in \mathcal{K}^{(n)}), \tag{6.3}$$

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 truncated distribution is non-zero. $\boldsymbol{\Theta}$ denotes the set of all model parameters. By leveraging truncated posteriors, the variational EM algorithm substantially reduces the computational complexity of both the E-step and M-step. However, the use of truncated posteriors requires a different implementation strategy compared to conventional EM to achieve practical speed-ups.

For instance, when computing squared Euclidean distances as in Eq. (6.2), instead of calculating the squared distance between each data point and the mean of every component, we only compute these distances between each data point and the components specified in the respective $\mathcal{K}^{(n)}$ sets. Consequently, instead of performing a full matrix multiplication between every row of $\boldsymbol{X}$ and every column of $\boldsymbol{\mu}^T$, we restrict the multiplication to only those columns of $\boldsymbol{\mu}^T$ that correspond to the indices in the respective $\mathcal{K}^{(n)}$ set. Furthermore,

[6] The squares are understood element-wise. Note that $\boldsymbol{X}^2 \mathbb{1}_{DC}$ can be computed more efficiently by first computing $\boldsymbol{X}^2 \mathbb{1}_D$, where $\mathbb{1}_D$ is D-dimensional vector of ones, and then broadcasting the result over C columns. The same applies for $\mathbb{1}_{ND}(\boldsymbol{\mu}^2)^T$.

the indices of columns in $\boldsymbol{\mu}^T$ are likely to differ for each row of $\boldsymbol{X}$. Figure 6.1 illustrates this procedure as an example.

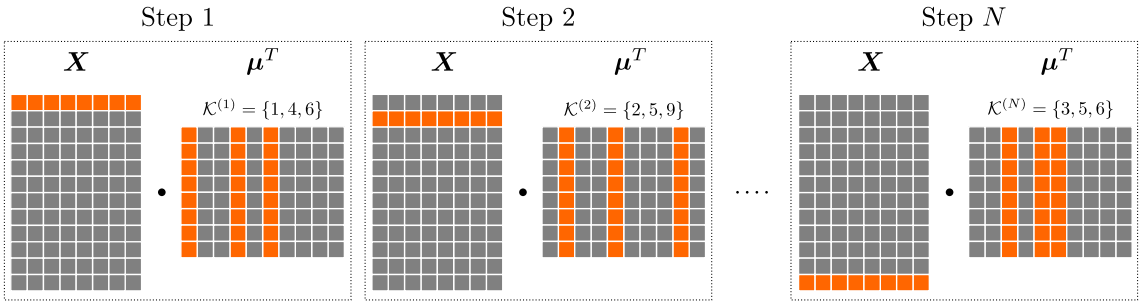

Fig 6.1: Matrix multiplication with sparse access pattern. Each row of $\boldsymbol{X}$ is multiplied with different columns of $\boldsymbol{\mu}^T$, as specified by the set $\mathcal{K}^{(n)}$. This operation is commonly required in the variational E-step of the variational EM algorithm.

Due to this non-uniform and irregular access patterns, where different rows of $\boldsymbol{X}$ are multiplied with distinct columns from $\boldsymbol{\mu}^T$, standard matrix multiplication techniques are not applicable for achieving practical speed-ups. Moreover, applying optimization strategies used in matrix multiplications, such as loop blocking, which increases hardware utilization, is highly non-trivial and may result in little to no speed-ups for variational EM. Consequently, the conventional EM algorithm can leverage hardware resources more efficiently than the variational EM algorithm. In practice and with currently available hardware, any implementation of the variational EM algorithm is expected to be less time-efficient *per joint computation* as the conventional EM algorithm (compare Fig. 6.2). Furthermore, the optimization using variational EM requires auxiliary computations not needed in the conventional EM algorithm. Consequently, if C' approaches C, which inherently occurs for small values of C, the conventional EM algorithm may exhibit shorter runtimes than the variational EM algorithm. However, for increasing number of components C and a fixed, relatively small value of C', the optimization using variational EM requires significantly fewer computations overall and can achieve substantially greater time-efficiency (compare Hirschberger *et al.*, 2022; Salwig *et al.*, 2025).

6.3 Implementation of the Variational EM Algorithm

As discussed in Sec. 6.2, commonly used Python libraries like NumPy and PyTorch are not well-suited for implementing the variational EM algorithm.

Given the sparse nature of truncated posteriors, it may seem natural to utilize sparse matrix libraries such as `scipy.sparse` from SciPy. The variational E-step requires combining two dense matrices into a sparse result following a sparse access pattern, as outlined in Sec. 6.2. However, SciPy does not offer such high-level function that supports this operation. Furthermore, the current version of SciPy lacks parallelized sparse matrix operations, which limits its efficiency for large-scale computations.

Notably, PyTorch is developing a sparse matrix sublibrary that includes the function `torch.sparse.sampled_addmm`, which can be leveraged to compute the variational E-step. However, this sublibrary is still in beta, and our initial tests suggest that the performance of `torch.sparse.sampled_addmm` remains suboptimal.

Given the lack of an efficient, parallelizable Python library for the variational E-step, we implemented the core functionalities of the variational EM algorithm in C++. Implementing these core functionalities in Python would result in significantly slower performance due to Python's nature as an interpreted language. However, given Python's user-friendly API, we opted to use Python as a wrapper to enhance the accessibility and usability of the variational EM implementation.

To provide details on the implementation of the variational EM algorithm, we discuss in the following the software libraries employed, the data structures for variational parameters, the implementation of the variational E-step and M-step, and challenges associated with parallelizing the algorithm.

6.3.1 Libraries

The following libraries are used in our implementation. In C++ we utilize the C++ STL (Standard Template Library, International Organization of Standardization (ISO), 2025) as well as the libraries Eigen (Guennebaud *et al.*, 2010), Boost (The Boost organization, 2024), pybind11 (Jakob *et al.*, 2017) and OpenMP (OpenMP Architecture Review Board, 2015). In Python, we utilize NumPy (Harris *et al.*, 2020).

Eigen is a header-only template library for linear algebra, including matrices, vectors and related algorithms. We mainly use Eigen to efficiently utilize the hardware when evaluating matrix and vector computations, such as dot products, due to Eigens lazy evaluation mechanism, vectorized operations, etc. All model parameters are stored as Eigen vectors or matrices.

Boost is a large collection of peer-reviewed C++ libraries. We only use a subset of Boost for its hash tables `boost::unordered_flat_set` and `boost::unordered_flat_map`, introduced in version 1.81.0. These hash tables provide substantially better performance compared to the C++ STL in terms of execution time.

The header-only library pybind11 enables data communication between C++ and Python. This library allows us to have the computationally expensive parts in C++, while having a convenient interface in Python. Importantly, pybind11 provides transparent conversion between Eigen matrices and NumPy arrays. Given the correct memory layout, no copy is required for using a dataset loaded as a NumPy array inside C++ or accessing the model parameters within Python.

OpenMP is uses for shared memory multiprocessing, as detailed in Sec. 6.3.5.

6.3.2 Data Structures for Variational Parameters

The variational EM algorithm introduces the variational distributions $q_n(c; \boldsymbol{\Theta})$ with additional variational parameters, including the $\mathcal{K}^{(n)}$ sets, the sets g_c, and the search spaces $\mathcal{S}^{(n)}$, all of which need to be memorized using suitable data structures.

To avoid storing redundant information, we do not explicitly store $\mathcal{K}^{(n)}$ sets or search spaces $\mathcal{S}^{(n)}$. Instead, these sets are implicitly given by the data structure holding the variational distributions. Specifically, for given indices n and c, the presence of a value for $q_n(c; \boldsymbol{\Theta})$ implies that c is in $\mathcal{K}^{(n)}$ or $\mathcal{S}^{(n)}$. As a result, $q_n(c; \boldsymbol{\Theta})$, $\mathcal{K}^{(n)}$, and $\mathcal{S}^{(n)}$ are all represented within the same data structure.

A natural choice in line with the truncated variational distributions would be to utilize sparse matrix representations, such as compressed row storage (CRS) or compressed column storage (CCS) formats. In these representations, the (n, c)-th element of the matrix corresponds to $q_n(c; \boldsymbol{\Theta})$, but we only store the non-zero values of the variational distributions. However, Eigen's implementations of these formats do not support parallelization, as their storage layouts prevent independent modification of individual rows or columns within a sparse matrix.

Instead of storing the variational distributions in a single sparse matrix, an alternative approach is to use a sparse vector for each $q_n(c; \boldsymbol{\Theta})$. This has the advantage that the vectors are independent of another, allowing for parallel processing. However, inserting new elements remains challenging. Random insertion into a sparse vector is $\mathcal{O}(nnz)$ where nnz denotes the number of non-zero elements in the vector. In contrast, inserting elements with increasing indices can be performed in $\mathcal{O}(1)$ (provided that sufficient memory is preallocated). However, this approach would require sorting the elements by there component index, introducing additional computational overhead.

Consequently, we did not use any predefined sparse matrix format for the time being, but instead employed a dynamic-sized array (a C++ STL `std::vector`) to represent each $q_n(c; \boldsymbol{\Theta})$, resulting in N such arrays. Each array stores pairs consisting the component index c and the value of $q_n(c; \boldsymbol{\Theta})$ for each c in $\mathcal{K}^{(n)}$ or $\mathcal{S}^{(n)}$. In this sense, there are similar to sparse vectors, but ordering of the indices is not required. Since the size of the search space is bounded, sufficient memory can be preallocated to prevent reallocations. Consequently, as no ordering is required, new elements can always be appended at the end of the array in $\mathcal{O}(1)$ complexity.

Another option would be to use hash maps, where the component indices c are the keys and the values correspond to $q_n(c; \boldsymbol{\Theta})$. However, using such hash tables to store the variational distributions proved to be disadvantageous due to the overhead associated with iterating over all elements. This inefficiency is particularly relevant given that the variational distributions must be traversed multiple times throughout the algorithm. In contrast, arrays offer a performance advantage since their elements are stored contiguously in memory, enabling faster iteration. We nonetheless utilize hash tables in intermediate steps, as detailed below.

The implementation of the sets g_c follow a similar strategy as the implementation of the variational distributions. The sets g_c are implemented using `std::vector`. For each component c, we maintain an array containing the relative components and c itself. Explicitly including c in g_c was a design decision to simplify the construction of the search spaces.

6.3.3 Variational E-step

Construct search spaces $\mathcal{S}^{(n)}$: Given the variational parameters $\mathcal{K}^{(n)}$ and g_c, the first step of the variational E-step involves constructing the search spaces $\mathcal{S}^{(n)}$ for each data point $\boldsymbol{x}_n$. This is done by expanding $\mathcal{K}^{(n)}$ into $\mathcal{S}^{(n)}$ by successively inserting all elements of all g_c with c in $\mathcal{K}^{(n)}$. Additionally, we uniformly sample one random component c and insert it into the search space.

To prevent duplicates in the search spaces, we use hash sets (`boost::unordered_flat_set`) to track already inserted indices. This dual bookkeeping ensures that all indices are unique in an efficient and effective manner.

```
for n = 1 : N do
    S^(n) = ⋃_{c∈K^(n)} g_c;
    S^(n) = S^(n) ∪ {c} with c ~ U{1, C};
```

Compute $\log p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta})$**:** We use log-joints rather than joints, as they offer greater numerical stability. The log prevents values from approaching zero, avoiding the problem of numerical underflow for small probabilities. Given the non-uniform and irregular access patterns (as discussed in Sec. 6.2), the computation of the log-joints $\log p(c, \boldsymbol{x}_n \mid \boldsymbol{\Theta})$ is evaluated using two nested loops over n and c. Within these loops, the log-joints are computed as defined by the generative model. Matrix and vector operations, such as dot products, are handled by Eigen to enable efficient, vectorized computations. Finally, the computed log-joint value is inserted into the data structure of $q_n(c; \boldsymbol{\Theta})$.

For the two nested loops, there are two possible iteration strategies. The first is to iterate over all data points $\boldsymbol{x}_n$ in the outer loop, and over all c in $\mathcal{S}^{(n)}$ in the inner (fast-running) loop (loop order $n \to c$). This is a natural choice, since for each data point $\boldsymbol{x}_n$, we store its corresponding components c, allowing for straightforward iteration over each n and its corresponding c in $\mathcal{S}^{(n)}$:

```
for n = 1 : N do
    for c ∈ S^(n) do
        q_n(c; Θ) = log p(c, x_n | Θ);
```

An alternative approach is to iterate over all components in the outer loop, and over all n considered for component c in the inner loop (loop order $c \to n$). However, adopting this loop order requires restructuring the storage format to associate each component with its corresponding data points. To obtain this storage order, we construct an array $\mathcal{S}_{(c)}$ for each component c, containing all data point indices n for which c is in $\mathcal{S}^{(n)}$. Following this, we can iterate over all components c and the corresponding data point indices $n \in \mathcal{S}_{(c)}$ to compute the log-joints:

```
for n = 1 : N do
    for c ∈ S^(n) do
        S_(c) = S_(c) ∪ {n}
for c = 1 : C do
    for n ∈ S_(c) do
        q_n(c; Θ) = log p(c, x_n | Θ);
```

While constructing $\mathcal{S}_{(c)}$ adds computational overhead, this reordering makes n the inner loop index and reduces the frequency of loading the model parameters into the cache. In particular, for models with a large number of parameters per component, the second approach may offer improved efficiency by minimizing redundant memory accesses and optimizing cache utilization (compare Tab. 6.1).

Update sets $\mathcal{K}^{(n)}$**:** After computing the log-joints, we select the C' components with the largest log-joint values for each data point to update the $\mathcal{K}^{(n)}$ sets. To achieve this, we

employ a partial sorting algorithm using `std::nth_element` to ensure that the C' elements with the largest log-joints occupy the first C' positions in the data structure of $q_n(c;\, \boldsymbol{\Theta})$. Since our implementation does not require a strict ordering of these elements, partial sorting is sufficient.

---

for $n = 1 : N$ **do**
  partially sort $q_n(c;\, \boldsymbol{\Theta})$ using `std::nth_element`$(q_n(c;\, \boldsymbol{\Theta}),\, C')$;
  $\mathcal{K}^{(n)}$ is given by the C' first elements of $q_n(c;\, \boldsymbol{\Theta})$;

An alternative approach for this selection would be to use heaps. However, we have refrained from using heaps, as it would require copying or moving the data into and out of the heap.

Update sets g_c: The sets g_c are updated based on a KL-Divergence approximation, defined as follows (for more details, see Salwig *et al.* (2025)):

$$D_{c\tilde{c}} = \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log \frac{p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})}{p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})} \delta(\tilde{c} \in \mathcal{S}^{(n)}) \quad \text{where} \quad N_{c\tilde{c}} = \sum_{n \in \mathcal{I}_c} \delta(\tilde{c} \in \mathcal{S}^{(n)}) \tag{6.4}$$
$$\text{and} \quad \mathcal{I}_c = \{n \mid c = c_n\} \quad \text{with} \quad c_n = \operatorname*{argmax}_{c' \in \mathcal{K}^{(n)}} p(c', \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$$

In our implementation, we store only the log-joints $\log p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})$ rather than the log-probabilities $\log p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})$. To utilize the log-joints, we reformulate Eq. (6.4) as follows:

$$D_{c\tilde{c}} = \log p(\tilde{c} \,|\, \boldsymbol{\Theta}) - \log p(c \,|\, \boldsymbol{\Theta}) + \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \left(\log p(c, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) - \log p(\tilde{c}, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})\right) \delta(\tilde{c} \in \mathcal{S}^{(n)}) \tag{6.5}$$

We begin by partitioning the dataset into C subsets $\mathcal{I}_c$. Concretely, for each component c, we construct an array $\mathcal{I}_c$ that stores the indices n of all data points for which c has the highest log-joint probability.

---

for $n = 1 : N$ **do**
  $c_n = \operatorname*{argmax}_{c \in \mathcal{K}^{(n)}} q_n(c;\, \boldsymbol{\Theta})$;
  $\mathcal{I}_{c_n} = \mathcal{I}_{c_n} \cup \{n\}$;

To compute then $D_{c\tilde{c}}$ for a given component c, we utilize two temporary data structures. The first is an array, where each entry consists of three elements: a component index $\tilde{c}$, a counter $N_{c\tilde{c}}$ tracking the frequency of a component $\tilde{c}$, and $D_{c\tilde{c}}$, the current sum of the log-joint differences $q_n(c;\, \boldsymbol{\Theta}) - q_n(\tilde{c};\, \boldsymbol{\Theta})$.

Additionally, we utilize a hash map (`boost::unordered_flat_map`) to efficiently keep track of the component indices $\tilde{c}$. The keys in this hash map are the indices $\tilde{c}$, while the values store their positions in the temporary array. This structure enables rapid lookup to determine whether $\tilde{c}$ is already present in the array.

For a given component c, we iterate then over all n in the partition $\mathcal{I}_c$ and all indices $\tilde{c}$ in the corresponding search spaces $\mathcal{S}^{(n)}$. During this process, we update $D_{c\tilde{c}}$ by adding the log-joint difference $q_n(c;\, \boldsymbol{\Theta}) - q_n(\tilde{c};\, \boldsymbol{\Theta})$, and incrementing the counter $N_{c\tilde{c}}$. If $\tilde{c}$ is not already present in the array, a new entry is appended; otherwise, its stored position is

retrieved from the map. In the pseudocode below, the entries in the set $\mathcal{D}_c$ represent the keys in the hash map.

---

for $c = 1 : C$ **do**
  $D_{c\tilde{c}} = 0.0$;
  $N_{c\tilde{c}} = 0$;
  $\mathcal{D}_c = \emptyset$;
  for $n \in \mathcal{I}_c$ **do**
    for $\tilde{c} \in \mathcal{S}^{(n)} \setminus \{c\}$ **do**
      $D_{c\tilde{c}}$ += $q_n(c;\, \boldsymbol{\Theta}) - q_n(\tilde{c};\, \boldsymbol{\Theta})$;
      $N_{c\tilde{c}}$ += 1;
      $\mathcal{D}_c = \mathcal{D}_c \cup \{\tilde{c}\}$;

To obtain the final estimate of the KL-Divergence, we normalize the values $D_{c\tilde{c}}$ by the counter $N_{c\tilde{c}}$ and add the difference between the log-priors. The corresponding loop over $\mathcal{D}_c$ is efficiently performed by simply iterating over all entries in the temporary array.

---

for $c = 1 : C$ **do**
  for $\tilde{c} \in \mathcal{D}_c$ **do**
    $D_{c\tilde{c}}$ /= $N_{c\tilde{c}}$;
    $D_{c\tilde{c}}$ += $\log p(\tilde{c} \,|\, \boldsymbol{\Theta}) - \log p(c \,|\, \boldsymbol{\Theta})$;

After all KL-Divergences for component c are estimated, we select the $G-1$ components with the smallest values to update g_c. Therefore, we again employ `std::nth_element`, ensuring that the $G-1$ smallest elements occupy the first $G-1$ positions in the array. Then, we use the indices $\tilde{c}$ of these first $G-1$ entries in addition to c as the new set g_c (as a strict ordering is not required).

---

for $c = 1 : C$ **do**
  partially sort $D_{c\tilde{c}}$ w.r.t. $\tilde{c}$ using `std::nth_element`($D_{c\tilde{c}}$, $G-1$);
  g_c is given by the $G-1$ first elements of $D_{c\tilde{c}}$ and c;

Once all sets g_c have been updated and the search spaces are no longer required, we discard all elements beyond the first C' from each $q_n(c;\, \boldsymbol{\Theta})$.

Compute posteriors: To compute the truncated posteriors from the log-joints for each data point, all operations are performed in-place on the data structure of $q_n(c;\, \boldsymbol{\Theta})$. For improved numerical stability, we employ the log-sum-exp trick[7].

---

for $n = 1 : N$ **do**
  $A = \max(q_n(c;\, \boldsymbol{\Theta}))$;
  for $c \in \mathcal{K}^{(n)}$ **do**
    $q_n(c;\, \boldsymbol{\Theta}) = \exp(q_n(c;\, \boldsymbol{\Theta}) - A)$;
    Z_n += $q_n(c;\, \boldsymbol{\Theta})$;
  for $c \in \mathcal{K}^{(n)}$ **do**
    $q_n(c;\, \boldsymbol{\Theta})$ /= Z_n;

---

[7] `https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`

6.3.4 Variational M-step

For the variational M-step, the update equations are model specific, but generally require the evaluation of expectation values over truncated posteriors. As with the computations of the log-joints, there are two possible iteration strategies for the updates of the model parameters. The straightforward approach would be to iterate over each data point and over all c in $\mathcal{K}^{(n)}$ in the inner loop. However, in a parallel execution, this loop order may lead to race conditions, as multiple threads could simultaneously update the expectation values or model parameters for the same component.

To prevent race conditions and enable efficient parallel execution, we decided to iterate over all components in the outer loop, and over all n considered for component c in the inner loop (similar to the loop order $c \to n$ in the E-step for MFAs). Therefore, we first construct an array for each component c, containing all data point indices n for which c is in $\mathcal{K}^{(n)}$.

For an efficient computation, the update equations are designed such that only a single pass over the dataset is required. For instance, in the case of variance estimation in a diagonal GMM, this is achieved by utilizing $\mathbb{E}\left[(x_{nd} - \mu_{cd})^2\right] = \mathbb{E}\left[x_{nd}^2\right] - \mu_{cd}^2$. Similarly, the update equations for the MFA model can be formulated to require only one pass over the dataset, as detailed in the appendix of Salwig *et al.* (2025).

6.3.5 Parallelization

With continuous advancements in hardware, modern CPUs feature an increasing number of physical cores. As a result, an additional implementation challenge is the effective parallelization of computations to leverage these multi-core architectures.

We employ OpenMP to distribute the workload of loops across the cores of a single CPU compute node. A parallelization over multiple nodes is currently not available, but can be implemented as an additional layer on top of the current shared-memory parallelization (see Sec. 6.5).

For an efficient parallel execution, the outermost loop is parallelized whenever possible, and the number of required synchronizations is minimized by using thread-local temporary variables. To prevent nested parallelization, we disabled Eigen's parallelization, as only a subset of Eigen's operations support it.

In the computation of the log-joints with loop order $c \to n$, the update of g_c, and the M-step, we previously described how arrays are constructed to group data points according to their corresponding components. These arrays are important for a cache-friendly and effective parallel execution, and to avoid race-conditions.

However, constructing these arrays *in parallel* is non-trivial. In the current implementation, we address this by allocating C arrays for each parallel thread, rather than maintaining a single set of C arrays accessed by all threads. Instead of all threads sorting the data points within a shared set of arrays, each thread sorts a subset of the data points into its own set of arrays.

Since the data points associated with a given component c are divided into multiple arrays (one per thread), the loop over all these data points is decomposed into two nested loops: an outer loop over the arrays assigned to each thread, and an inner loop traversing over each data point within each of these arrays. As an example, the loop decomposition for

computing the log-joints with loop order $c \to n$ is displayed in the pseudocode below. Here, N_{threads} denotes the total number of threads.

```
for c = 1 : C do                          →   for c = 1 : C do
    for n ∈ S_(c) do                              for thread = 1 : N_threads do
        ⌙ ...                                         for n ∈ S_(c)[thread] do
                                                          ⌙ ...
```

A remaining challenge is that the number of elements in each of these arrays is initially unknown. While the number of data points per component is naturally bounded by N, preallocating a total of $N_{\text{threads}} \cdot N \cdot C$ elements is impractical for large scale applications. Consequently, dynamic memory reallocations may be necessary. However, since the allocated memory is reused in subsequent iterations, we expect that the arrays sizes will stabilize over time. As a result, most reallocations should occur in the first few iterations, minimizing the impact on the overall performance.

To avoid reallocations, we experimented with using lists (`std::list`) instead of arrays. Lists support both element appending and concatenation in $\mathcal{O}(1)$. However, iteration over all elements in the concatenated list proved to be significantly slower compared to using `std::vector`, as the elements are not aligned in memory.

6.4 Experiments

In the following, we numerically evaluate the performance of the variational EM algorithm applied to GMMs with diagonal covariances (v-GMM^d) and the MFA model (v-MFA). We omit a detailed comparison of the performance of conventional and variational EM, as this has already been thoroughly addressed in Salwig *et al.* (2025).

Numerical experiments are conducted on the Letter (Slate, 1991), EMNIST (Cohen *et al.*, 2017, using the "byclass" split), and CIFAR-10 (Krizhevsky, 2009) datasets, to cover a variety of data points sizes N and data dimensions D. To increase the amount of available data, we merged the official training and test splits. Experimental settings and hardware configurations follow those described in Salwig *et al.* (2025) unless otherwise stated. Concretely, across all datasets and experiments, we set $C = 1000$ and the variational hyperparameters to $C' = 3$ and $G = 15$.

Time of Log-Joint Evaluation in Conventional vs. Variational EM: Figure 6.2 shows the average wall-clock runtime of a single log-joint evaluation and an entire E-step. Although the variational EM algorithm consistently requires more time for a single log-joint evaluation, the overall time of an E-step is significantly shorter compared to the conventional EM algorithm.

Profiling of the variational E- and M-step: A detailed breakdown of the runtime of individual steps for the variational EM algorithm is presented in Fig. 6.3. The reported runtime for the variational E-step includes the time required to compute the log-joints and posteriors, excluding the auxiliary part, which is shown separately. The auxiliary part of the E-step comprises the time spent for the construction of the search spaces $\mathcal{S}^{(n)}$, and the updates of the sets g_c and $\mathcal{K}^{(n)}$, which are not required in the conventional EM algorithm. The variational M-step includes the runtime to compute the expectation values w.r.t. the posteriors and to update the model parameters. From Fig. 6.3, it can be seen that in most

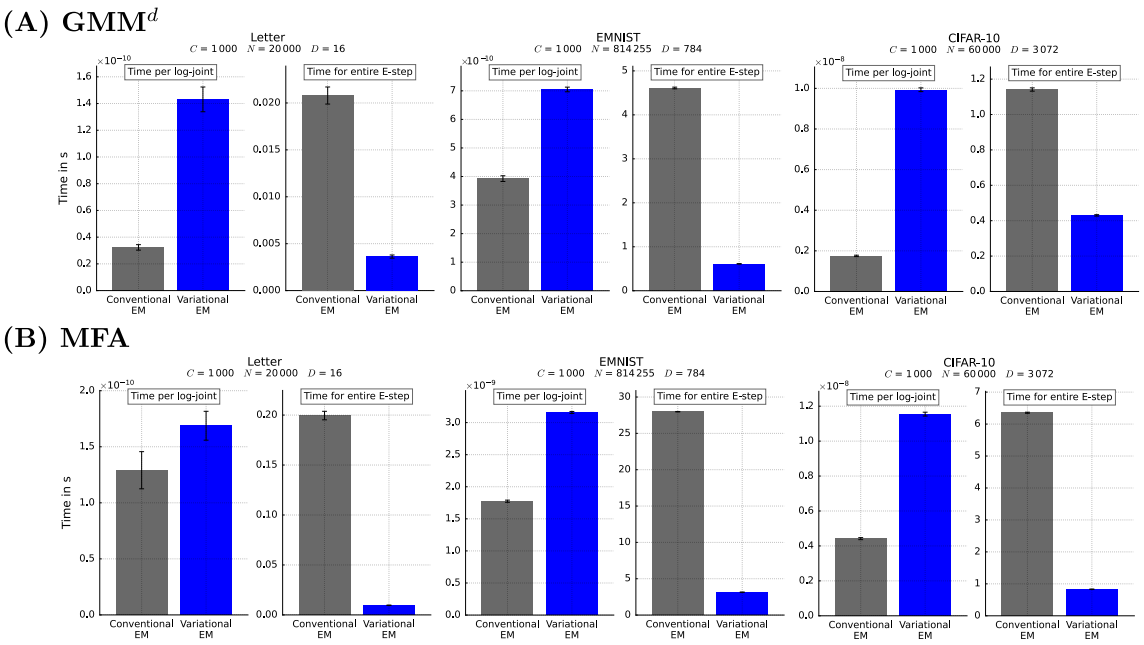

Fig 6.2: Average wall-clock runtime of a single log-joint evaluation and an entire E-step for both the conventional and variational EM algorithms. Each subplot corresponds to one dataset. **(A)** shows the results for a diagonal GMM, while **(B)** provides the corresponding measurements for an MFA model. All results are averaged across iterations and over 8 independent runs. Error bars denote standard errors of the mean.

cases, the longest runtime is required for computing log-joints. This step also represents the part of the algorithm with the highest theoretical complexity, especially for large values of N, C and D (see Salwig *et al.*, 2025). Only for v-GMM^d on the Letter dataset, the auxiliary part takes longer than the rest of the E-step. However, the overall runtime for Letter is small compared to EMNIST or CIFAR-10.

Loop Order: In Sec. 6.3.3, we discussed how different looping strategies for computing the log-joints can result in faster algorithms depending on the number of model parameters per component. In Tab. 6.1, we show runtime measurements of the different loop orders ($n \to c$ or $c \to n$) for both v-GMM^d and v-MFA.

From Tab. 6.1, it becomes evident that for v-GMM^d, the loop order $n \to c$ is more favorable. However, for v-MFA, the loop order $c \to n$ results in about 4 to 5× faster runtimes compared to $n \to c$ for EMNIST and CIFAR-10, despite the computational overhead of reordering the loop. Only on Letter, the loop order $c \to n$ results in longer runtimes, which may be due to the small data dimension D.

Parallelization: Figure 6.4 shows the runtime speed-ups compared to a serial execution of variational EM across different numbers of threads. Both v-GMM^d and v-MFA exhibit comparable speed-ups across the different datasets. For EMNIST and CIFAR-10, the observed speed-up factors scale near-linear, close to the theoretical maximum (dashed lines in Fig. 6.4). In general, the theoretical maximal speed-up can never be fully reached due to Amdahl's Law (see Bryant and O'Hallaron, 2011, Sec. 5.14.3), as well as startup and termination costs of parallel regions and synchronization overheads.

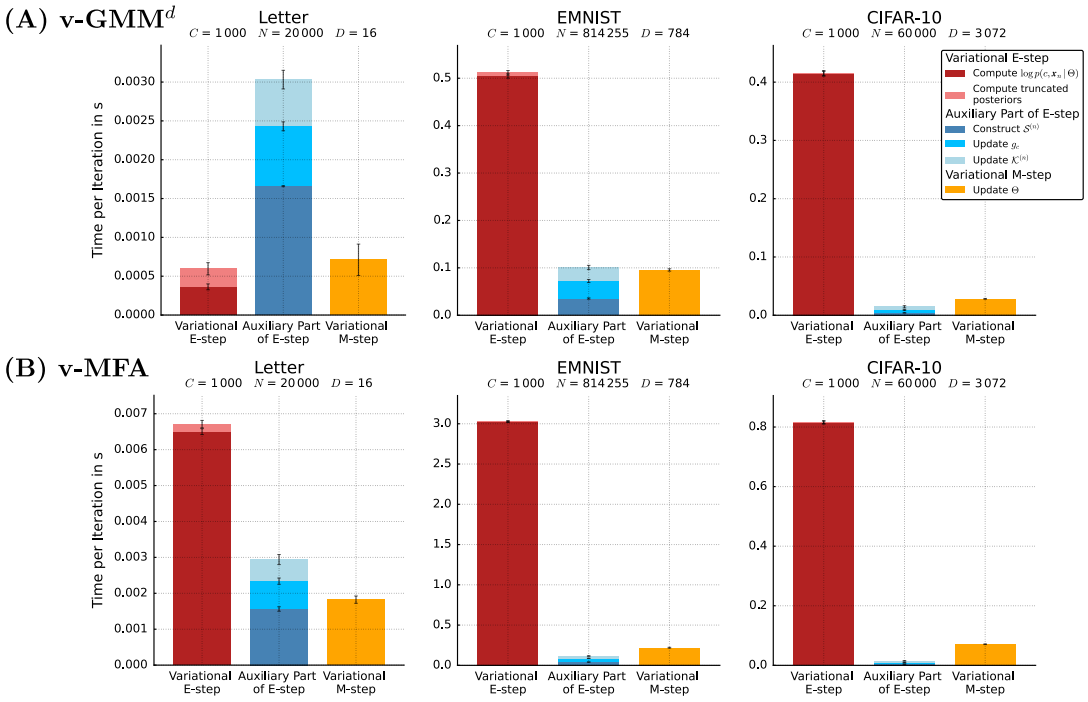

Fig 6.3: Average wall-clock runtimes of different steps in the variational EM algorithm. Each subplot corresponds to one dataset. **(A)** shows the results for v-GMM^d, while **(B)** provides the corresponding measurements for v-MFA. All results are averaged across iterations and over 8 independent runs. Error bars denote the sum of the standard errors of the means of the stacked bars. The legend holds for all subfigures.

For Letter, the parallelization performance is suboptimal, indicating that the variational EM algorithm can not fully utilize the additional hardware resources. This can be attributed to the relatively larger auxiliary part of the E-step (see Fig. 6.2), which appears to parallelize less efficiently. Additionally, the overall runtime on Letter is significantly shorter compared to EMNIST or CIFAR-10. Consequently, parallelization overheads become more pronounced relative to the total runtime.

Throughout all three datasets, the speed-up appears to plateau at 64 threads. This may be due to memory operations, suggesting that the execution becomes memory-bound at this point. As the number of threads increases to 64, memory bandwidth limitations likely hinder further performance gains.

Tab 6.1: Average wall-clock runtimes for computing log-joints in the variational EM algorithm for different loop orders for both v-GMM^d and v-MFA. Loop order $n \to c$ denotes that n is the outer loop and c is the inner (fast-running) loop. For $c \to n$, the order is reversed and c is the outer loop. All results are averaged across iterations and over 8 independent runs. Error bars denote standard errors of the mean.

Algorithm	Loop Order	Letter	EMNIST	CIFAR-10
v-GMM^d	$n \to c$	(0.36 ± 0.04) ms	(0.50 ± 0.00) s	(0.41 ± 0.00) s
	$c \to n$	(2.27 ± 0.05) ms	(2.87 ± 0.02) s	(0.63 ± 0.00) s
v-MFA	$n \to c$	(2.70 ± 0.09) ms	(14.65 ± 0.20) s	(4.63 ± 0.02) s
	$c \to n$	(6.50 ± 0.08) ms	(3.02 ± 0.01) s	(0.81 ± 0.00) s

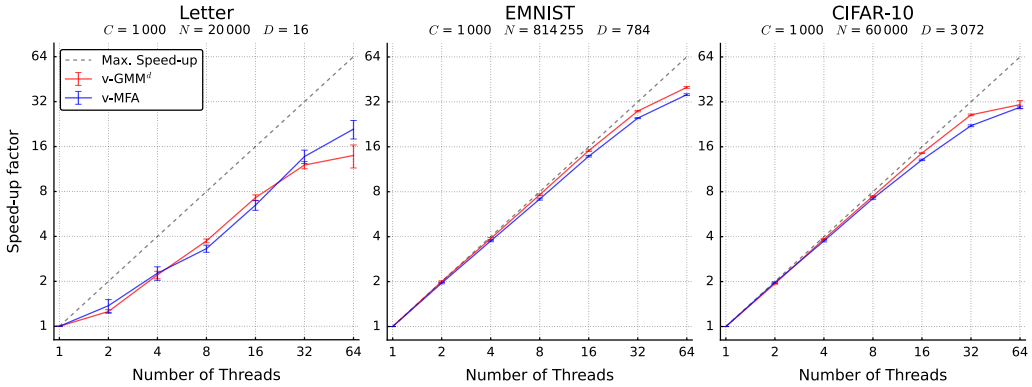

Fig 6.4: Parallelization speed-up of both v-GMM^d and v-MFA using different numbers of threads in comparison to a serial execution using one thread. The dashed line represents the theoretical maximal speed-up. Each subplot corresponds to one dataset. All results are averaged over 8 independent runs. Error bars denote the standard errors of the mean. The legend holds for all subfigures.

6.5 Discussion and Outlook

The variational EM algorithm examined in this technical report not only offers a theoretical reduction in complexity but can also be highly efficient in practice. It demonstrates higher efficiency compared to the conventional EM algorithm in practical applications (Hirschberger *et al.*, 2022; Salwig *et al.*, 2025). This increased efficiency can be attributed to the reduced computational requirements introduced by the truncated posteriors (see Eq. (6.3)). However, the use of truncated posteriors necessitated the adoption of an alternative implementation strategy compared to those employed by the conventional EM algorithm. Furthermore, the sparse nature of the truncated posteriors make the variational EM algorithm less time-efficient *per posterior computation* than the conventional EM algorithm, particularly with currently available hardware (compare Fig. 6.2). Nevertheless, for $C' \ll C$, the variational EM algorithm requires far fewer posterior computations than the conventional EM algorithm, resulting in much higher overall efficiency in practice. Conversely, when C' approaches C, the variational EM algorithm becomes less time-efficient than the conventional EM algorithm (even when all auxiliary functions are disabled). In a future version of the algorithm, strategies such as loop blocking or tiling could be explored, which are known to contribute to the high efficiency of the conventional EM algorithm. Concrete performance gains when using such approaches for the variational EM algorithm remain to be investigated.

Another critical factor influencing the efficiency of the variational EM algorithm is its parallelization. While the conventional EM algorithm is naturally well-suited for parallelization, the variational EM algorithm presented specific challenges in this regard (compare Sec. 6.3.5). For datasets with a small number of data points N and low dimensionality D (e.g., the Letter dataset), the parallelization of the variational EM algorithm is suboptimal, suggesting that it cannot fully leverage the hardware resources (see Fig. 6.4). However, for datasets with a large number of data points N and high dimensionality D, the parallelization of the variational EM scales nearly linearly. This may be attributed to the differing contributions to the overall runtime, with the dominant runtime components varying significantly between small and large datasets. For the Letter dataset, the runtime proportion of the auxiliary steps is relatively high, whereas for the other datasets, the

dominant portion of the runtime is spent on the computation of the log-joints. This suggests that the auxiliary steps are less effectively parallelized. To further optimize the algorithm for small-scale datasets, these auxiliary steps and their parallelization present opportunities for further improvements.

The current implementation of the variational EM algorithm supports parallelization only within a single CPU node. While this has been sufficient to optimize flexible GMMs with billions of parameters on datasets containing millions of data points — scales that have previously not been addressed with GMMs — it limits the algorithm's scalability and restricts its ability to handle still larger datasets. A key limiting factor in this context is the available memory within a single node. To overcome this memory constraint, the variational EM algorithm could be extended to parallelize across multiple CPU nodes using libraries such as MPI (Message Passing Interface, Message Passing Interface Forum, 2023) in the future.

To further enhance the runtimes of the variational EM algorithm, alternative hardware, such as GPUs, could be utilized. However, this would require to rewrite a major part of the code in a GPU programming language such as CUDA (Nickolls *et al.*, 2008). As discussed in Sec. 6.3, PyTorch is developing a sparse matrix library that may be suitable for implementing the variational EM algorithm, and provides GPU support. Although this library is still in beta, it could be a promising option for a future implementation of the variational EM algorithm.

Chapter 7

Discussion and Conclusion

Machine learning (ML) technologies have become increasingly integrated into everyday life, with millions of people relying on them for a wide range of applications, such as autocorrection in text messaging (e.g., Bryant *et al.*, 2023), route planning in navigation systems (e.g., Derrow-Pinion *et al.*, 2021) or translating text between languages (e.g., Bahdanau *et al.*, 2014), to illustrate just a few examples. Among the various ML approaches, deep learning (DL) models, i.e., models incorporating deep neural networks (DNNs), have emerged as one of the most widely adopted and extensively used approaches. A key factor for their widespread adoption is their scalability which has led to strong performance on several (especially supervised) tasks including image classification, natural language processing and speech recognition. A major research focus in the field of DL has been the development of unprecedented large-scale models, i.e., DNNs with parameter counts in the hundreds of billions, which are trained on massive datasets. In particular, large-scale deep generative models (DGMs; e.g., Brown *et al.*, 2020; Chowdhery *et al.*, 2023) have gained substantial attention due to their ability to generate novel and realistic data, including text, images or music. A contributing factor to the excessive scaling of DNNs has also been the relative ease of application and scalability of optimization methods used to train DNNs, such as stochastic gradient descent and its variants (e.g., Kingma and Ba, 2014). Although these optimization methods theoretically scale linearly w.r.t. the number of data points and model parameters, the scaling of the model sizes has resulted in an exponential increase in computational demands, energy consumption and associated carbon emissions (Cottier *et al.*, 2024).

As one of the major contributions of this thesis, we have derived and investigated a highly efficient alternative training technique that does not rely on gradient descent and is applicable to probabilistic generative models with categorical latents: the variational EM algorithm introduced in Ch. 3. Notably, the integration of this variational EM algorithm with mixtures of factor analyzers (MFAs) does not yield a fixed, constant speed-up; rather, the resulting v-MFA algorithm exhibits *sublinear* scaling w.r.t. the number of data points and model parameters (see Fig. 3.4). Concretely, we observed that the number of joint evaluations per data point scales with *exponents* consistently below 1/3 (see Sec. 3.3.1). In practical terms, this implies that even if the model size increases eightfold, the computational cost during training only approximately doubles. Due to this sublinear scaling, the v-MFA algorithm enables the novel training of very large-scale (non-DNN-based) representations, with remarkably low computational costs (see Fig. 3.6). Moreover, to the authors' knowledge,

this includes the training of the largest-scale Gaussian mixture models (GMMs) reported to date. Thereby, MFAs themselves provide efficient approximations that reduce the complexity of optimizing general GMMs in high-dimensional spaces by modeling the data within each mixture component along lower-dimensional manifolds (compare Sec. 3.2.1). Although v-MFA is a variational method, which typically involves trade-offs in approximation quality, empirical results show that any loss in performance is minimal or even negligible in practice (compare Figs. 3.5 and C.1).

The proposed variational EM algorithm falls within a broader class of variational methods that use truncated posteriors as their family of variational distributions and treat the discrete latent states as variational parameters, referred to as $\mathcal{K}^{(n)}$ (e.g., Drefs *et al.*, 2022; Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Lücke, 2019). These approaches share a common focus on computational efficiency, although they differ in the specific strategies used to optimize the variational parameters $\mathcal{K}^{(n)}$. Concretely, they vary in how they propose new latent states while avoiding the need to iterate through all possible states. The proposed algorithm is specifically tailored for the efficient optimization of probabilistic models with categorical latent variables, especially mixture models. Concretely, it aims to reduce the complexity w.r.t. the number of mixture components C. Among existing methods, it is most closely related to those of Hirschberger *et al.* (2022) and Exarchakis *et al.* (2022), which are applied to GMMs and use similarity measures between clusters (or components) to guide the selection of new candidate states. These approaches rely on either stochastic sampling methods (Exarchakis *et al.*, 2022) or on the use of datapoint-independent sets, where each set contains potential replacement candidates for a given latent state (Hirschberger *et al.*, 2022). However, previous theoretical treatments were based on approximations of Euclidean distances, which limit their generalizability to generative models beyond GMMs with isotropic (or diagonal) covariance structures. In the proposed method, we adopt the use of datapoint-independent sets, as in Hirschberger *et al.* (2022), to propose new candidate states. However, rather than relying on Euclidean distances, our derivation is based on approximations of the Kullback-Leibler (KL) divergence and does not depend on any model-specific assumptions (see Sec. 3.2.3). Importantly, this makes the overall variational EM algorithm generally applicable to generative models with categorical latent variables, including other types of mixture models (compare also Sec. 7.1). The optimization of the variational parameters is fully defined in terms of the joint distributions given by the model.

After deriving the variational EM algorithm proposed in Ch. 3, we investigated its application for optimizing probabilistic generative models (e.g., GMMs and MFAs) in both semi-supervised and unsupervised learning tasks. In addition to this variational EM approach, we also examined an alternative variational optimization technique based on truncated posteriors, namely, the evolutionary variational optimization (EVO) method (Drefs *et al.*, 2022). EVO integrates evolutionary algorithms into the variational framework to optimize the variational parameters $\mathcal{K}^{(n)}$ and has been shown to efficiently optimize various probabilistic data models with binary latent variables (e.g., Drefs *et al.*, 2022, 2023; Mousavi *et al.*, 2023). The main difference between the proposed variational EM algorithm and EVO lies in the types of latent variable models they are intended to optimize: the variational EM algorithm is specifically tailored to models with categorical latent variables, such as mixture models, whereas EVO is designed for multiple-cause models involving binary latent variables.

In Chs. 2 and 5, we investigated these variational optimization methods in blind zero-shot denoising, a fully unsupervised task. In this setting, the algorithm is provided with only a single noisy image, without access to any additional information (e.g., noise level) or to external training or ground-truth data. Algorithms that can be applied in such blind zero-shot scenarios are particularly valuable in domains where data acquisition is difficult or where the imaged structures are rare (e.g., in medicine, astronomy or microscopy). In such cases, relying on external training data, whose content statistics diverge substantially from those of the target image, can lead to undesirable effects, such as the introduction of artifacts or the suppression of important image details (Laine *et al.*, 2021; Shocher *et al.*, 2018).

As generative models, we explored Poisson mixture models (PMMs), spike-and-slab sparse coding (SSSC, in combination with EVO, referred to as ES3C), MFAs and GMMs which were optimized using the variational optimization methods (or the conventional EM algorithm in the case of PMMs). We compared these generative models against a variety of other efficient (blind) zero-shot denoising algorithms, including advanced filter-based methods (e.g., Azzari and Foi, 2016b; Dabov *et al.*, 2007; Makitalo and Foi, 2011) as well as a range of state-of-the-art DNN-based methods applicable in zero-shot denoising settings (e.g., Batson and Royer, 2019; Huang *et al.*, 2021; Krull *et al.*, 2019a; Lequyer *et al.*, 2022b; Prakash *et al.*, 2021b; Quan *et al.*, 2020a; Ulyanov *et al.*, 2018).

In contrast to the generative models, the comparison algorithms are specifically tailored and extensively tuned for the task of image denoising. For instance, the DNN-based approaches often incorporate image-specific inductive biases directly through their architectural design (e.g., via convolutional neural networks, which are well suited for capturing local patterns in spatially structured data). The generative models, however, are generic probabilistic models and do not encode any image-specific inductive biases and remain agnostic to the particular characteristics of image data.

Remarkably, even without such task-specific biases, these generative models produced denoising results that were highly competitive with those of the filter-based and DNN-based methods (compare Figs. 2.2 to 2.4 and Tab. 5.1). Nevertheless, in terms of peak performance (measured by PSNR and SSIM), the generative models were outperformed by certain specialized DNN-based approaches (e.g., Noise2Fast (N2F) in Fig. 2.4 and Self2Self (S2S) in Tab. 5.1). Importantly, these results also highlight the sensitivity of DNN-based methods (and also filter-based methods) to model assumptions and architectural choices. When such assumptions are misaligned with the statistical properties of the data, performance can degrade substantially (as observed for N2F and Noise2Void (N2V) in Fig. 2.2 or S2S producing blurry images in Figs. 2.2 to 2.4). In contrast, the results of the generative models, e.g., PMMs and ES3C, exhibited more robust performance across all datasets, even on data for which their noise assumptions were presumably not met (compare PMM in Figs. 2.3 and 2.4).

When comparing the runtimes (see Tab. A.2 and Tab. 5.1), substantial variations (ranging from seconds to days) were observed across the various algorithms. The fastest methods were the filter-based approaches, which do not involve any learning process. However, other efficient methods included N2F and mixture model-based approaches (e.g., GMMs and MFAs) when optimized using the proposed variational EM algorithm. In contrast, the DNN-based method S2S (which achieved the best denoising performance in several settings, e.g., Tab. 5.1) frequently required substantially longer training times (e.g., in the order of days for a single image). Notably, the v-MFA algorithm proved to be a highly competitive

method (in terms of denoising performance), while also yielding very short runtimes (cf. Tab. 5.1).

We further investigated the use of GMMs for denoising positron emission tomography (PET) images under zero-shot conditions (see Voskamp et al., 2025 in Publication List). The results demonstrated that the GMM-based denoising approach can potentially reduce the amount of radioactivity required in activation studies, particularly in investigations of the auditory system, without significantly compromising diagnostic quality. Moreover, it has recently been argued that the use of DL models in PET imaging can present significant challenges (Hellwig *et al.*, 2023). These challenges stem from inherent limitations, such as their black-box nature in scenarios where interpretability is essential, and the need for high-quality training data, which can be difficult to obtain in domains like medical imaging. In contrast, elementary data models such as PMMs and GMMs are typically characterized by a small, explicit set of assumptions about the data. This makes them more transparent and interpretable than complex models like DNNs, and involves substantially fewer design choices compared to DL methods. Taken together, these observations suggest that such elementary probabilistic models, when integrated with the efficient optimization techniques investigated here, may provide valuable approaches for denoising applications in domains such as PET imaging.

As a further major contribution of this thesis, we have integrated the variational optimization techniques (introduced in Ch. 4) into a graph-based semi-supervised learning (GSSL) framework. Specifically, we focused on a class of GSSL methods characterized by first learning a distribution model over the data, followed by graph construction and label propagation performed on the learned representations (e.g., Liu *et al.*, 2010; Wang *et al.*, 2016; Zhang *et al.*, 2023). These distribution-based GSSL methods address the high computational complexity of conventional GSSL approaches, which operate directly on the data points and typically scale with $\mathcal{O}(N^3)$, by reducing it to a linear dependency on the number of data points N. However, this efficiency gain comes at the cost of learning a distribution model, which often becomes the new computational bottleneck.

With the integration of variational optimization techniques into the distribution-based GSSL framework, we were not only able to accelerate the distribution learning stage, but also to inherently induce the construction of sparse graphs, which in turn improved the efficiency of the graph construction stage. In contrast, previous approaches constructed sparse graphs in an *ad hoc* manner (Liu *et al.*, 2010; Qiu *et al.*, 2019; Wang *et al.*, 2016). We combined this novel GSSL framework with GMMs using diagonal covariance matrices as well as with MFAs, referred to as v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL, respectively. While diagonal GMMs have previously been applied in the context of distribution-based GSSL (e.g., Zhang *et al.*, 2023), MFAs, to our knowledge, have not yet been explored in this setting. A fundamental assumption in SSL is the *manifold assumption*, which posits that high-dimensional data lie near a (lower-)dimensional manifold (compare, e.g., van Engelen and Hoos, 2020). This assumption is inherently well captured by the MFA model, making it particularly well suited for use in SSL.

The results in Sec. 4.3 reveal not only substantial runtime improvements of v-GMM$^{\mathrm{d}}$-GL over previous state-of-the-art approaches (see Fig. 4.4), but also that both v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL achieve higher classification performance across several benchmark datasets (see Fig. 4.5 and Tab. C.2). Notably, v-MFA-GL outperformed existing methods by a substantial margin on some benchmarks, which may be attributed to its ability to incorporate the manifold structure of the data into its representations. Moreover, the results highlight

that variational optimization methods enable the practical use of more expressive mixture models, such as MFAs, within the GSSL framework, which would otherwise be impractical due to their long optimization times (compare the runtimes of v-MFA-GL and MFA-GL in Fig. 4.4).

The graph construction in this novel GSSL approach is closely related to the guidance of new candidate states (for optimizing the variational parameters $\mathcal{K}^{(n)}$) in the variational EM algorithm introduced in Ch. 3. In the current formulation, the selection of new candidate states is guided by an approximation of the KL-divergence between mixture components (compare Sec. 3.2.3). This procedure can be interpreted as constructing a directed graph over the mixture components, where each set g_c defines the nodes associated with component c, and the approximate KL-divergence values serve as edge weights in the graph. Such a graph could, for instance, be used to construct the Laplacian matrix (cf. Eq. (4.8)) within the GSSL framework – or, vice versa, the graph structure exploited in GSSL (cf. Eq. (4.5)) could inform the candidate selection process in the variational EM algorithm (in future work).

A common challenge in SSL (as well as in unsupervised learning) is the tuning of model hyperparameters, which can be critical for achieving strong performance. Due to the limited number of labeled data points, reliable hyperparameter optimization is often infeasible under realistic SSL conditions. Consequently, many studies rely on additional validation sets containing substantially more labeled data. However, this practice is inconsistent with the assumptions of SSL and has been explicitly criticized in prior work (e.g., Forster *et al.*, 2018; Oliver *et al.*, 2018). To maintain a fair and realistic evaluation protocol, we used default hyperparameter values for all compared algorithms across all datasets. Additionally, the hyperparameter analysis presented in Fig. C.2 indicates that both v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL are robust across a wide range of hyperparameter values, but the performance could be further improved through dataset-specific hyperparameter tuning.

Moreover, we adopted in Ch. 4 an existing GSSL benchmark proposed in prior work (Zhang *et al.*, 2023), which includes specific preprocessing steps for the datasets. To further assess the impact of these preprocessing steps, we conducted additional experiments and observed that the performance of the evaluated algorithms is indeed influenced by the preprocessing pipeline (compare Appendix C.2.5). This effect is particularly pronounced for the CIFAR-10 dataset, where different preprocessing strategies yield markedly different results. In particular, applying the algorithms to feature vectors extracted from a pretrained ResNet model leads to significantly better performance compared to using raw input data or applying only PCA (compare Fig. C.4). Therefore, it is important to note that the CIFAR-10 results from Ch. 4 may not be directly comparable to those reported in the literature, as the strong performance of all algorithms is largely attributable to the pretrained ResNet model, which was trained supervised on a much larger labeled dataset. Nonetheless, within the scope of the adopted GSSL benchmark, the comparison between algorithms remains fair, as all algorithms are applied to the same ResNet-derived feature representations.

In this thesis, we consistently examined semi-supervised and unsupervised learning tasks in which the algorithms were applied directly to the same data they were intended to process or infer from. Specifically, we focused on two settings: (i) the unsupervised zero-shot denoising setting, where the algorithm is trained directly on the data it aims to denoise, and (ii) the label propagation setting, where the distribution models are trained on the data from which they are expected to infer labels. The proposed variational EM algorithm is particularly well-suited for these settings, as the variational parameters $\mathcal{K}^{(n)}$ can be

efficiently optimized during training and subsequently reused for downstream tasks such as denoising or semi-supervised classification. However, when the task involves inference on previously unseen data (without training), the variational EM algorithm may become inefficient. In contrast to conventional EM, which typically performs only a single E-step (or a neural network, which requires just a single forward pass), variational EM requires the optimization of the variational parameters for each new data point, i.e., optimizing the variational parameters $\mathcal{K}^{(n)}$ in repeated E-steps (note, however, that in very large-scale models, even a single full E-step may be computationally infeasible, whereas repeated variational partial E-steps might still be tractable). Moreover, it remains to be investigated whether the quality of inference from repeated E-steps on unseen data matches that achieved when both the variational parameters and the model parameters are jointly optimized during training.

In several experiments presented in this thesis, we compared the efficiency of the proposed algorithms (e.g., v-MFA, v-MFA-GL or v-GMM$^{\mathrm{d}}$-GL) with benchmark algorithms in terms of wall-clock runtime (compare, e.g., Figs. 3.5 and 4.4). This metric is particularly relevant for assessing practical applicability. However, it should be noted that absolute runtime values are inherently affected by implementation details and the underlying hardware used in the experiments. To ensure a fair comparison, we placed special emphasis not only on implementing the proposed algorithms efficiently (cf. Ch. 6) but also on (re-)implementing the baseline methods with comparable care (compare Appendix B.2.1.1 and Appendix C.1.3) which often resulted in faster runtimes than those of the original implementations. To this end, we focused on developing lightweight and user-friendly Python frameworks (often with core functionality implemented in C++), and we made the software for both our own algorithms and the comparison methods readily available to researchers interested in applying them.

7.1 Outlook

Building on the results of this thesis, and partly based on discussions among the authors of Salwig *et al.* (2025), a variety of future research directions exists. One potential direction is the further development of the variational EM algorithm itself. Concretely, improvements could focus on enhancing the proposal of latent states to enable still more effective and efficient optimization of the variational parameters. For instance, graph construction techniques introduced in Ch. 4 could be adapted, or the variational EM algorithm could be integrated with sampling-based methods (e.g., Exarchakis *et al.*, 2022). Moreover, it could be investigated how the training principles of the variational EM algorithm might be adapted to, or inspire, the training algorithms of other generative models including, e.g., multiple-cause models or DGMs. One possible approach would be to integrate the variational EM algorithm with other variational optimization techniques (e.g, EVO, Drefs *et al.*, 2022). To enable efficient and fast inference on unseen data (without training), investigating amortized variational distributions also represents a possible research direction (Ganguly *et al.*, 2023).

Another promising direction is to integrate the variational EM algorithm with alternative classes of mixture models or probabilistic models with categorical latents. This may include, for instance, mixture models from the exponential family (Banerjee *et al.*, 2005), mixture of experts (Jordan and Jacobs, 1994), hidden Markov models (Murphy, 2022) or also the well-known k-means algorithm, which can be seen as an approximated and constrained

GMM optimization (Forster and Lücke, 2018). Furthermore, the variational optimization approach could be employed to train more expressive mixture models that explicitly capture data-specific biases modeled by discrete latents (Dai *et al.*, 2013; Dai and Lücke, 2014; Drgas *et al.*, 2017; Lücke *et al.*, 2009; Monk *et al.*, 2018) including translation invariance or object order in depth. For instance, such models, when combined with efficient training techniques, may also be particularly well suited for image enhancement tasks such as denoising.

Arguably the most promising direction for future work lies in exploring the scalability of the v-MFA algorithm. The novel ability to scale flexible GMMs (or other generative models in future work) with only minor loss in performance can not only enhance efficiency in existing applications but also can open the possibility of applying these models to tasks and domains where they were previously replaced or disregarded due to scalability limitations. For instance, a fundamental task of semi-supervised and unsupervised learning is density estimation, and GMMs are particularly well-suited for this task, as they can approximate arbitrary probability distributions in $\mathbb{R}^D$ with increasing accuracy as the number of components grows (e.g., Li and Barron, 1999; Maz'ya and Schmidt, 1996; Zeevi and Meir, 1997). Therefore, the proposed v-MFA algorithm may provide a scalable and effective approach for high-quality density estimation, even in high-dimensional spaces (compare, e.g., Richardson and Weiss, 2018). Moreover, it could be investigated whether probabilistic data models such as GMMs also exhibit performance-related scaling laws w.r.t. the model size or data size, as recently demonstrated for large-scale models incorporating DNNs (Henighan *et al.*, 2020; Kaplan *et al.*, 2020).

The promising results presented in Chs. 2, 4 and 5 further motivate the exploration of the v-MFA algorithm (along with potentially more expressive mixture models in the future) for tasks such as semi-supervised classification and denoising. In this context, the scalability of the v-MFA algorithm could be exploited to enable its application to larger datasets. This could be achieved, for example, through data augmentation techniques to train larger models, which have the potential to improve classification performance (as preliminary evidence from MNIST8M suggests, compare Appendix C.2.4). Similarly, the v-MFA algorithm could be investigated in non-zero-shot denoising settings where training is performed on a large volume of noisy (or clean) images.

To further accelerate the variational EM algorithm (whose current implementation supports parallelization only on a single CPU node, see Ch. 6), a potential direction for future research is to extend the algorithm to enable distributed parallelization across multiple CPU nodes or to adapt it for execution on alternative hardware, such as GPUs (also compare Sec. 6.5).

Appendix A

Supplementary Information on Zero-Shot Denoising of Microscopy Images Recorded at High-Resolution Limits

A.1 Details on Denoising with PMMs

In order to enhance a microscopy image with PMMs, we extract overlapping patches (Burger *et al.*, 2012; Elad and Aharon, 2006; Mairal *et al.*, 2008; Zhou *et al.*, 2012) from the image and treat the obtained set of image patches as our training dataset $\mathcal{X} = \{\boldsymbol{x}_n\}_{n=1}^N$ (patch sizes used are listed in Appendix A.2). Given this training dataset, we seek parameters $\boldsymbol{\Theta}^* = \text{argmax}_{\boldsymbol{\Theta}}\, \mathcal{L}(\boldsymbol{\Theta})$ that optimize the data log-likelihood. A direct optimization is usually challenging, and, instead, efficient algorithms use, for instance, Expectation Maximization approaches (Neal and Hinton, 1998; Saul and Jordan, 1995) and optimize a lower bound (also known as free energy or evidence lower bound – ELBO)

$$\mathcal{F}(q, \boldsymbol{\Theta}) \leq \mathcal{L}(\boldsymbol{\Theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}_n \mid \boldsymbol{\Theta}) = \sum_{n=1}^{N} \log \sum_{c=1}^{C} p(\boldsymbol{x} \mid c, \boldsymbol{\Theta}) p\left(c \mid \boldsymbol{\Theta}\right) \tag{A.1}$$

of the log-likelihood. $p\left(c \mid \boldsymbol{\Theta}\right)$ and $p\left(\boldsymbol{x} \mid c, \boldsymbol{\Theta}\right)$ are referred as prior and noise model, respectively, given by

$$p\left(c \mid \boldsymbol{\Theta}\right) = \pi_c, \qquad \text{and} \qquad p\left(\boldsymbol{x} \mid c, \boldsymbol{\Theta}\right) = \prod_{d=1}^{D} \text{Poiss}\left(x_d; \mu_{cd}\right), \tag{A.2}$$

where $\boldsymbol{\pi} = \{\pi_c\}_{c=1}^C$ denote the components' prior activations, for which $\pi_c \in [0, 1]$ and $\sum_{c=1}^C \pi_c = 1$ applies; $\boldsymbol{\mu} = \{\boldsymbol{\mu}_c\}_{c=1}^C$ denote cluster centers with $\boldsymbol{\mu}_c = \{\mu_{cd}\}_{d=1}^D$ and C denotes the number of cluster components. $\mathcal{F}(q, \boldsymbol{\Theta})$ is iteratively and alternatingly optimized w.r.t. the variational distributions $q = \{q_n\}_{n=1}^N$ (E-step) and the model parameters $\boldsymbol{\Theta}$ (M-step). In order to turn the inequality in Eq. (A.1) into an equality such that the lower bound matches the likelihood, the functionals q_n are chosen to match the exact posterior in the E-step:

$$p(c \mid \boldsymbol{x}, \boldsymbol{\Theta}) = \frac{p(\boldsymbol{x} \mid c, \boldsymbol{\Theta})\, p\left(c \mid \boldsymbol{\Theta}\right)}{\sum_{c'=1}^{C} p(\boldsymbol{x} \mid c', \boldsymbol{\Theta})\, p(c' \mid \boldsymbol{\Theta})}. \tag{A.3}$$

In the M-step, the parameters μ_{cd} and π_c are updated using the following update equations

$$\mu_{cd} = \frac{\sum_{n=1}^{N} p(c \mid \boldsymbol{x}_n, \boldsymbol{\Theta})\, x_d^{(n)}}{\sum_{n=1}^{N} p(c \mid \boldsymbol{x}_n, \boldsymbol{\Theta})}, \qquad \pi_c = \frac{1}{N} \sum_{n=1}^{N} p(c \mid \boldsymbol{x}_n, \boldsymbol{\Theta}). \tag{A.4}$$

Given a set of optimized parameters $\boldsymbol{\Theta}$, we can use the learned data representation under the PMM model for probabilistic data estimation: for instance, we can estimate the most likely, non-noisy version $\boldsymbol{x}^{\,\mathrm{est}}$ of a given noisy data point $\boldsymbol{x}$ by estimating the first moment of the modeled pixel distribution. Here, we adapt the line of reasoning used in Drefs *et al.* (2022), and derive a data estimator based on the posterior predictive distribution $p(\boldsymbol{x}^{\,\mathrm{est}} \mid \boldsymbol{x}, \boldsymbol{\Theta})$. For the PMM of Eq. (A.2), the posterior predictive distribution can be written as follows:

$$p(\boldsymbol{x}^{\,\mathrm{est}} \mid \boldsymbol{x}, \boldsymbol{\Theta}) = \sum_{c} p(\boldsymbol{x}^{\,\mathrm{est}} \mid c, \boldsymbol{\Theta})\, p(c \mid \boldsymbol{x}, \boldsymbol{\Theta}). \tag{A.5}$$

The noisy pixel values are then replaced by the expectation values of this posterior predictive distribution:

$$x_d^{\mathrm{est}} \leftarrow \mathbb{E}_{p(\boldsymbol{x}^{\,\mathrm{est}} \mid \boldsymbol{x}, \boldsymbol{\Theta})}\big[x_d^{\mathrm{est}}\big] = \mathbb{E}_{p(c \mid \boldsymbol{x}, \boldsymbol{\Theta})}\big[\mathbb{E}_{p(\boldsymbol{x}^{\,\mathrm{est}} \mid c, \boldsymbol{\Theta})}\big[x_d^{\mathrm{est}}\big]\big], \tag{A.6}$$

where $\mathbb{E}$ denotes expectation and $\mathbb{E}_{p(x)}\big[g(x)\big] = \sum_x p(x) g(x)$ for discrete x with $\sum_x$ running over all possible configurations of x. The inner expectation is the first moment of the noise model (Eq. (A.2), right). This expectation is given by $\mathbb{E}_{p(\boldsymbol{x}^{\,\mathrm{est}} \mid c, \boldsymbol{\Theta})}\big[x_d^{\mathrm{est}}\big] = \mu_{cd}$. Inserting this into Eq. (A.6), the data estimator yields to:

$$\mathbb{E}_{p(\boldsymbol{x}^{\,\mathrm{est}} \mid \boldsymbol{x}, \boldsymbol{\Theta})}\big[x_d^{\mathrm{est}}\big] = \mathbb{E}_{p(c \mid \boldsymbol{x}, \boldsymbol{\Theta})}\big[\mu_{cd}\big]. \tag{A.7}$$

We apply Eq. (A.7) to each data point in $\mathcal{X}$ (i.e., to each image patch). Due to mutually overlapping patches, this results in multiple denoised estimates for a given image pixel, and, here, we consider median values of pixel estimations for image reconstruction (see Fig. A.1 for an illustration of the pipeline). Note that this pipeline is not specific to PMM but can be adopted to other generative models (compare Drefs *et al.*, 2022, the here reported results of ES3C were obtained with the same procedure).

A.2 Details on Numerical Experiments

A.2.1 Data

Tab. A.1 lists the file names of the nine TEM Images of SARS-CoV-2 infected cell cultures that we selected for our evaluations. Fig. A.2 depicts the signal and background regions used for the signal-to-noise quantification in Fig. 2.2B.

Implementations and hyperparameters

The following list gives details about the source codes used for the algorithms we compare. All used software sources are publicly available except of PMM, which is an implementation of a standard Poisson mixture model made available together with this paper. Following the respective instructions, it was straightforward to obtain, install, and execute the algorithms using relatively standard software setups (Linux distributions, python, MATLAB, etc.).

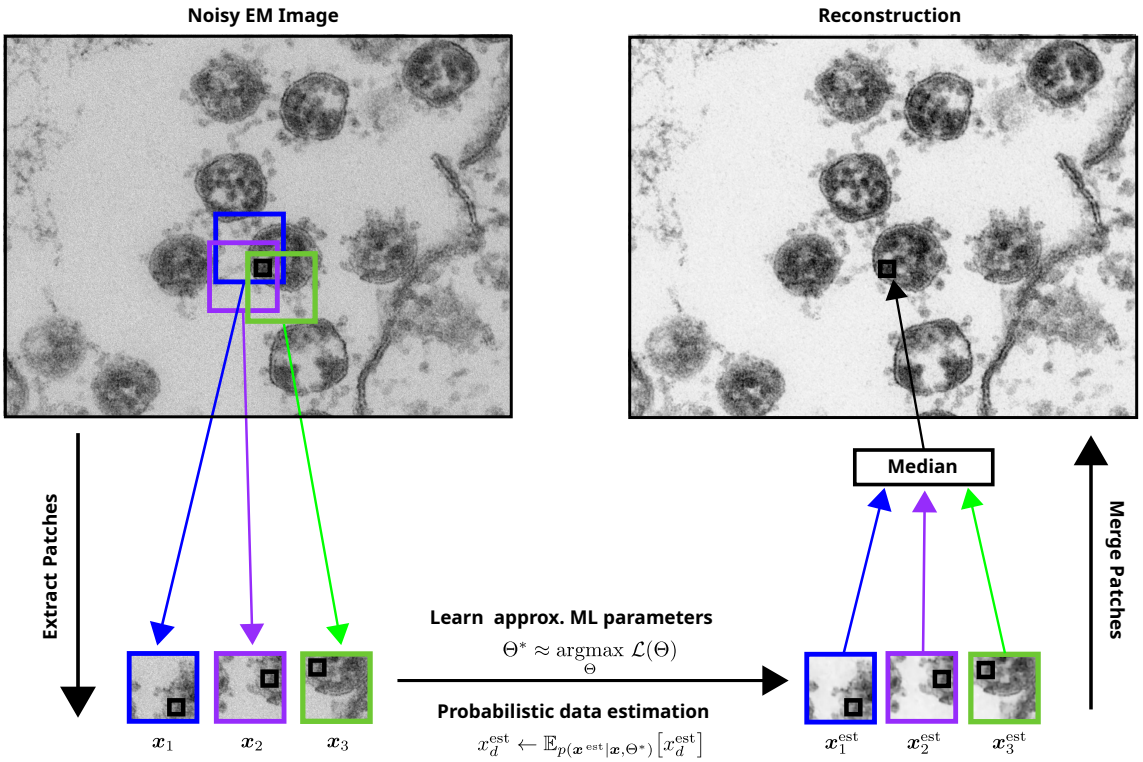

Fig A.1: Illustration of patch-based blind zero-shot image denoising using probabilistic generative models (compare Methods in the main text): As input (top left), we use a noisy image (for instance, a TEM image of SARS-CoV-2 infected cell cultures). Using patches extracted from the noisy image (bottom left), we first learn a probabilistic representation of the image using an appropriately chosen data model (e.g., ES3C or PMM). We can then apply the learned representation to probabilistically reconstruct each image patch (bottom right). Finally, we generate a reconstructed image by computing median values of pixel estimates obtained from mutually overlapping patches (top center and right). The image enhancement approach does not require (clean) training data and can directly be applied to a single noisy image.

Median Filtering. We used the implementation of a median filter provided by the SciPy (ndimage) package (Virtanen *et al.*, 2020).

BM3D. We used the official BM3D Python software (Mäkinen, 2007) and executed the algorithm using the configuration 'stage_arg=BM3DStages.ALL_STAGES'. To provide a noise level estimate to the algorithm, we used the method of Chen *et al.* (2015a) and the respective publicly available implementation (Chen *et al.*, 2015b) together with default hyperparameters.

VST+BM3D. We used the official VST+BM3D software (Mäkitalo and Foi, 2011) and executed the algorithm with the configuration of the 'demo_Poisson_experiments_table' example available in the file package.

I+VST+BM3D. We used the official I+VST+BM3D software (Azzari and Foi, 2016a) and executed the algorithm with the configuration of the 'demo_iterVSTpoisson' example available in the file package.

ID	Name
1	Dataset_02_SARS-CoV-2_007.tif
2	Dataset_02_SARS-CoV-2_009.tif
3	Dataset_02_SARS-CoV-2_038.tif
4	Dataset_03_SARS-CoV-2_043.tif
5	Dataset_03_SARS-CoV-2_070.tif
6	Dataset_03_SARS-CoV-2_080.tif
7	Dataset_07_SARS-CoV-2_036.tif
8	Dataset_07_SARS-CoV-2_077.tif
9	Dataset_07_SARS-CoV-2_102.tif

Tab A.1: File names of the selected TEM images of SARS-CoV-2 infected cell cultures (compare Datasets in the main text).

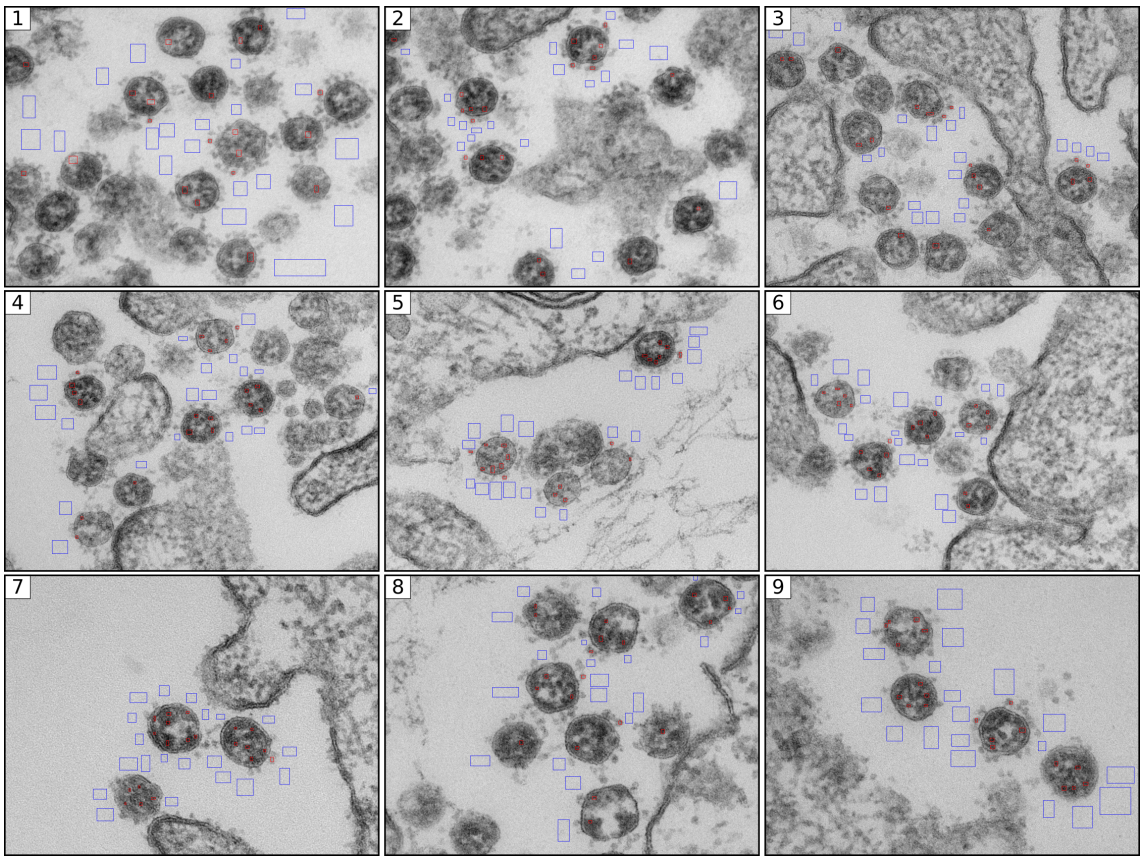

Fig A.2: Illustration of the selected TEM images of SARS-CoV-2 infected cell cultures and hand-labeled signal (red) and background (blue) regions. For each TEM image, 20 signal and background regions were labeled, respectively (compare Evaluation Metrics in the main text).

N2V. We used the official Noise2Void implementation (Krull *et al.*, 2019b) and executed the algorithm with the configuration of the 'BSD68_reproducibility' example available in the respective GitHub repository.

DivN. 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 the Mouse Actin, Cilia and SARS-CoV-2 experiments, we adopted the settings of the Convallaria example. To create a noise model, we used the bootstrapping method based on the respective denoised outcome of the N2V algorithm (following the example code).

S2S. We used the official Self2Self implementation (Quan *et al.*, 2020b) and executed the algorithm with the configuration of the 'demo_denoising' script available in the respective GitHub repository. We set the dropout rate to 0.3 (as suggested in examples in the 'demo_denoising' script), and the parameters 'sigma' and 'is_realnoisy' to '-1' and 'True', respectively. We modified the preprocessing pipeline by dividing the pixel amplitudes with the maximum pixel amplitude of a given input image rather than with the value 255 (compare 'utils.py/ line 26'). Accordingly, we modified the postprocessing pipeline by multiplying the pixel amplitudes of the output image with the previously determined maximum value and then saved the image with single precision (compare 'demo_denoising.py/ line 56').

N2F. We used the official Noise2Fast implementation (Lequyer *et al.*, 2022a) and executed the algorithm with the configuration of the 'N2F' script available in the respective GitHub repository.

ES3C. We used the official ES3C implementation (EVO developers, 2022) and adapted the 'image-denoising' example available in the respective GitHub repository for our purposes. Hyperparameters were set as follows: 'patch_height=6', 'H=512', 'Ksize=30', 'parent_selection=fit', 'mutation_algorithm=randflip', 'no_parents=20', 'no_children=1', 'no_generations=1', 'no_epochs=20'. We executed the implementation in parallel on 60 Intel Xeon Platinum 9242 CPUs.

PMM. We used our own implementation (Salwig, 2024) of a standard Poisson mixture model trained with Expectation Maximization, and data estimation analogous to ES3C for denoising (see Appendix A.1 for details). To train PMM on the microscopy images, we used a patch size and a cluster size of $D = 6 \times 6$ (same value as used for ES3C) and $C = 1000$, respectively. PMM was trained for 100 epochs for each image. We initialized PMM as follows: Priors $\{\pi_c^{\text{init}}\}_{c=1}^{C}$ were uniformly randomly drawn from the interval $(0, 1)$. The columns of the matrix W^{init} were initialized with the centers found by running the kmeans++ algorithm on the patch data. We executed the implementation in parallel on an Intel Xeon 4214 CPU.

Runtimes

Tab. A.2 lists the hardware we used to execute the implementations of each algorithm and provides an overview of their approximate runtimes in our experiments.

Acknowledgments. We would like to thank Michael Laue for discussions, support and for his very useful suggestions.

Tab A.2: Hardware and approximate runtime used to execute the implementations of the investigated algorithms on the considered datasets. The runtimes were not measured with the aim of enabling a systematic comparison, but to indicate an approximate order of magnitude.

Algorithm	Dataset			Hardware	
	SARS-CoV-2	Cilia	Convallaria	Device	Model name
Median Filtering	< 1 s	$\approx$ 6s	< 1 s	CPU	Intel Xeon E5-1620 v2
BM3D	$\approx$ 37 s	$\approx$ 106 s	$\approx$ 27 s	CPU	Intel Xeon E5-1620 v2
VST+BM3D	$\approx$ 31 s	$\approx$ 107 s	$\approx$ 34 s	CPU	Intel Xeon E5-1620 v2
I+VST+BM3D	$\approx$ 38 s	$\approx$ 191 s	$\approx$ 34 s	CPU	Intel Xeon E5-1620 v2
N2V	$\approx$ 3 h	$\approx$ 3 h	$\approx$ 3 h	GPU	NVIDIA Tesla V100 SXM2 (32GB)
DivN	$\approx$ 15 min	$\approx$ 11 min	$\approx$ 11 min	GPU	NVIDIA Tesla V100 SXM2 (32GB)
S2S	$\approx$ 13 h	$\approx$ 39 h	$\approx$ 10 h	GPU	NVIDIA Tesla V100 SXM2 (32GB)
N2F	$\approx$ 131 s	$\approx$ 8 min	$\approx$ 89 s	GPU	NVIDIA Tesla V100 SXM2 (32GB)
ES3C	$\approx$ 18 min	$\approx$ 46 min	$\approx$ 14 min	CPU	60$\times$ Intel Xeon Platinum 9242
PMM	$\approx$ 10 min	$\approx$ 50 min	$\approx$ 11 min	CPU	Intel Xeon 4214

Algorithm	Dataset			
	Cilia	Convallaria	Mouse Nuclei	Mouse Actin
Noisy	0.18	0.88	0.78	0.56
Median	0.59	0.82	0.87	0.82
BM3D	0.70	**0.97**	**0.96**	0.88
VST+BM3D	0.28	0.89	0.79	0.63
I+VST+BM3D	0.27	0.89	0.79	0.63
N2V	0.63	**0.97**	**0.96**	0.84
DivN	0.61	**0.97**	**0.96**	0.88
S2S	**0.75**	0.91	0.92	0.82
N2F	0.59	**0.97**	**0.96**	0.90
ES3C	0.67	0.96	**0.96**	**0.92**
PMM	0.42	0.96	0.95	0.78

Tab A.3: Structural similarity index (SSIM) values for the denoised TEM images of Cilia and the FM images of Convallaria, Mouse Nuclei and Mouse Actin of Figs. 2.2 to 2.4 in the main text. For the stochastic approaches (algorithms listed below, including, N2V in the table), we performed three independent executions of each experiment and here list averages over these runs; standard deviations were smaller or equal 0.01. The bold numbers denote the best SSIM value for each dataset.

	Convallaria	Mouse Nuclei	Mouse Actin
$\text{DivN}_{\text{C}}^{1}$	38.81 ± 0.06	36.11 ± 0.36	35.06 ± 0.25
$\text{DivN}_{\text{B}}^{1}$	38.46 ± 0.23	35.96 ± 0.20	35.12 ± 0.17
$\text{DivN}_{\text{C}}^{\text{all}}$	39.65 ± 0.02	37.32 ± 0.12	35.82 ± 0.01
$\text{DivN}_{\text{B}}^{\text{all}}$	39.05 ± 0.45	37.09 ± 0.06	35.80 ± 0.01

Tab A.4: PSNR values (in dB) obtained with different variants of DivN for the considered fluorescence microscopy images: For the results of Fig. 2.4 in the main text, we executed DivN using N2V-based bootstrapping for noise model estimation and a single noisy image for training (compare Methods in the main text). In further control experiments, we also investigated applications of DivN with calibration data for noise model estimation and training on full image series. For calibration, we used publicly available calibration images (Krull *et al.*, 2020a; Prakash *et al.*, 2019a,b). In total, we performed four different types of experiments per image, which we here refer to as $\text{DivN}_{\text{C}}^{1}$, $\text{DivN}_{\text{B}}^{1}$, $\text{DivN}_{\text{C}}^{\text{all}}$, and $\text{DivN}_{\text{B}}^{\text{all}}$. The superscripts ^{1} and $^{\text{all}}$ denote the variants of the algorithm that use training on a single and on all images of a given dataset, respectively; the subscripts $_{\text{C}}$ and $_{\text{B}}$ indicate noise model estimation using calibration data and bootstrapping, respectively. The table lists averages and standard deviations over three executions of the algorithm per setting.

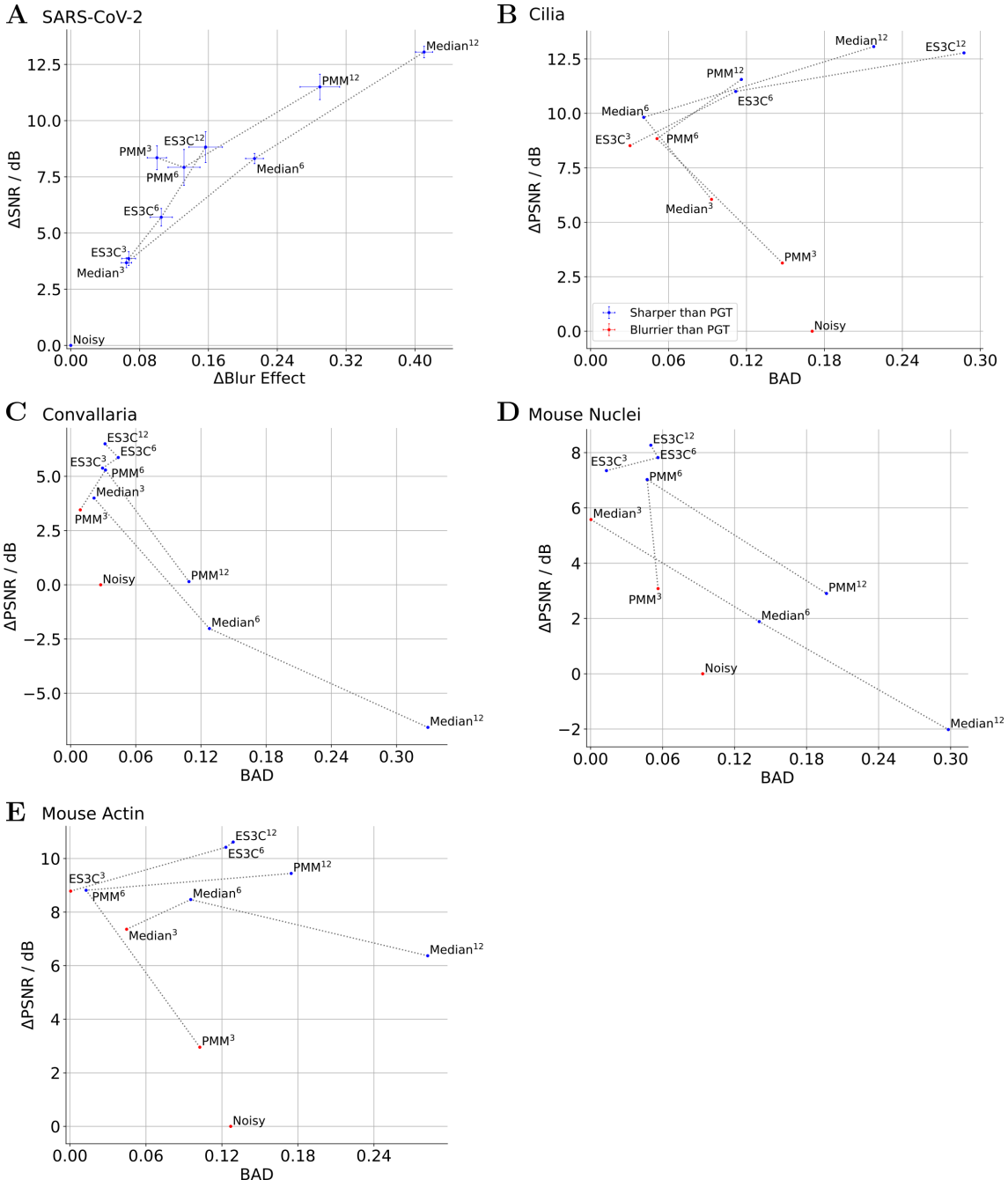

Fig A.3: Influence of the patch size hyperparameter on the performance of median filtering, ES3C and PMM. Compared are results for three different patch sizes (3×3, 6×6 and 12×12) for the investigated datasets. Noise suppression and preservation of image sharpness are quantified analogously as in Figs. 2.2 to 2.4. In subplot A, SNR and blur effect values correspond to averages and standard errors of the mean (SEM) over the nine considered test images. In subplots B–E, PSNR and BAD values of PMM and ES3C correspond to averages and SEM values of three independent runs of the algorithm (note that the SEM values are so small that they are hardly visible). The trend of the algorithms across the three different patch sizes is represented by the dashed lines.

Appendix B

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

B.1 Supplementary Information on Methods

In the following, we provide further details on techniques to avoid inversions of large covariance matrices in Appendix B.1.1, the derivation of the M-step parameter updates in Appendix B.1.2, the proof of Proposition 1 in Appendix B.1.3, additional information on the estimation of component-to-component distances and the KL-divergence in Appendix B.1.4, as well as further details on the algorithmic complexity of the variational E-step in Appendix B.1.5.

B.1.1 Avoiding Inversion of Large Covariance Matrices in MFAs

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

$$\begin{aligned}\log p(\boldsymbol{x}_n, c \,|\, \boldsymbol{\Theta}) &= \log p(c \,|\, \boldsymbol{\Theta}) + \log p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta}) \\ &= \log \pi_c - \tfrac{D}{2}\log(2\pi) - \tfrac{1}{2}\log|\boldsymbol{\Sigma}_c| - \tfrac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_c), \end{aligned} \tag{B.1}$$

the crucial aspects include the evaluation of the squared Mahalanobis distance $(\boldsymbol{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_c)$, which necessitates inverting $\boldsymbol{\Sigma}_c$, and the determinant computation for $\log|\boldsymbol{\Sigma}_c|$. Direct calculations involving $\boldsymbol{\Sigma}_c$ 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 $\boldsymbol{\Sigma}_c$, we utilize Woodbury's matrix inversion lemma

$$\boldsymbol{\Sigma}_c^{-1} = \boldsymbol{D}_c^{-1} - \boldsymbol{D}_c^{-1}\boldsymbol{\Lambda}_c \left(\boldsymbol{I} + \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1}\boldsymbol{\Lambda}_c\right)^{-1} \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} = \boldsymbol{D}_c^{-1} - \boldsymbol{U}_c\boldsymbol{V}_c \tag{B.2}$$

where we define

$$\boldsymbol{U}_c := \boldsymbol{D}_c^{-1}\boldsymbol{\Lambda}_c \in \mathbb{R}^{D\times H}, \quad \boldsymbol{L}_c := \boldsymbol{I} + \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1}\boldsymbol{\Lambda}_c = \boldsymbol{I} + \boldsymbol{U}_c^\top \boldsymbol{\Lambda}_c \in \mathbb{R}^{H\times H}, \tag{B.3}$$

$$\boldsymbol{V}_c := \boldsymbol{L}_c^{-1}\boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} = \boldsymbol{L}_c^{-1}\boldsymbol{U}_c^\top \in \mathbb{R}^{H\times D}. \tag{B.4}$$

Given that $\boldsymbol{D}_c$ is diagonal, its inversion is trivial. Furthermore, the inversion of the matrix $\boldsymbol{L}_c$ 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 Eq. (B.2) that the squared Mahalanobis distance, expressed as

$$\boldsymbol{v}^\top \boldsymbol{\Sigma}_c^{-1}\boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{D}_c^{-1}\boldsymbol{v} - (\boldsymbol{v}^\top \boldsymbol{U}_c)(\boldsymbol{V}_c\boldsymbol{v}), \tag{B.5}$$

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

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

$$\log|\boldsymbol{\Sigma}_c| = \log\left|\boldsymbol{\Lambda}_c\boldsymbol{\Lambda}_c^\top + \boldsymbol{D}_c\right| = \log\left|\boldsymbol{I} + \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1}\boldsymbol{\Lambda}_c\right| + \log|\boldsymbol{D}_c| = \log|\boldsymbol{L}_c| + \sum_d \log\sigma_{cd}^2.$$

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 B.1.2. 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.

B.1.2 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 Eq. (3.1) 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 $\boldsymbol{D}_c$ for each component, we present here the complete derivations for clarity and completeness.

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

$$p(c\,|\,\boldsymbol{\Theta}) = \pi_c, \quad p(\boldsymbol{z}\,|\,\boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{z};\boldsymbol{0},\boldsymbol{I}), \quad p(\boldsymbol{x}\,|\,\boldsymbol{z},c,\boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x};\boldsymbol{\Lambda}_c\boldsymbol{z}+\boldsymbol{\mu}_c,\boldsymbol{D}_c). \tag{B.6}$$

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

$$\begin{aligned} q(c,\boldsymbol{z};\boldsymbol{x}_n,\mathcal{K}^{(n)},\tilde{\boldsymbol{\Theta}}) &= \frac{1}{Z_n}p(c,\boldsymbol{z}\,|\,\boldsymbol{x}_n,\tilde{\boldsymbol{\Theta}})\delta(c\in\mathcal{K}^{(n)}) \\ &= \frac{1}{Z_n}p(c\,|\,\boldsymbol{x}_n,\tilde{\boldsymbol{\Theta}})p(\boldsymbol{z}\,|\,c,\boldsymbol{x}_n,\tilde{\boldsymbol{\Theta}})\delta(c\in\mathcal{K}^{(n)}) \\ &= q_n(c;\,\tilde{\boldsymbol{\Theta}})q_{n,c}(\boldsymbol{z};\,\tilde{\boldsymbol{\Theta}}), \end{aligned} \tag{B.7}$$

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

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

The distribution $q_n(c;\,\tilde{\boldsymbol{\Theta}})$ mirrors Eq. (3.4) in the main text, consistent with previous work employing truncated variational approximations for mixture models (e.g., Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022). We recognize that $q_{n,c}(\boldsymbol{z};\,\tilde{\boldsymbol{\Theta}}) = p(\boldsymbol{z} \,|\, c, \boldsymbol{x}_n, \tilde{\boldsymbol{\Theta}})$ conforms to a Gaussian distribution $\mathcal{N}(\boldsymbol{z}; \tilde{\boldsymbol{\mu}}_z, \tilde{\boldsymbol{\Sigma}}_z)$, with

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

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

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

where $\mathcal{H}\left[q\right] = -\sum_n \mathbb{E}_q[\log q(c, \boldsymbol{z} \,|\, \boldsymbol{x}_n, \tilde{\boldsymbol{\Theta}})]$ denotes the Shannon entropy. Substituting Eq. (B.7) into the free energy yields

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

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

$$\begin{aligned}
&\mathbb{E}_{q_{n,c}}\left[\log p(c, \boldsymbol{z}, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta})\right] = \mathbb{E}_{q_{n,c}}\left[\log p(c \,|\, \boldsymbol{\Theta}) + \log p(\boldsymbol{z} \,|\, \boldsymbol{\Theta}) + \log p(\boldsymbol{x}_n \,|\, c, \boldsymbol{z}, \boldsymbol{\Theta})\right] \\
&\quad = \log \pi_c - \tfrac{D}{2} \log(2\pi) + \tfrac{1}{2} \log |\boldsymbol{D}_c^{-1}| - \tfrac{1}{2} \mathbb{E}_{q_{n,c}}\left[(\boldsymbol{x}_n - \boldsymbol{\mu}_c - \boldsymbol{\Lambda}_c \boldsymbol{z})^\top \boldsymbol{D}_c^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_c - \boldsymbol{\Lambda}_c \boldsymbol{z})\right] \\
&\quad = \log \pi_c - \tfrac{D}{2} \log(2\pi) + \tfrac{1}{2} \log |\boldsymbol{D}_c^{-1}| - \tfrac{1}{2} \mathbb{E}_{q_{n,c}}\left[(\boldsymbol{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\boldsymbol{z}})^\top \boldsymbol{D}_c^{-1} (\boldsymbol{x}_n - \hat{\boldsymbol{\Lambda}}_c \hat{\boldsymbol{z}})\right] \qquad \text{(B.8)}
\end{aligned}$$

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

$$\hat{\boldsymbol{\Lambda}}_c := \begin{bmatrix} \boldsymbol{\Lambda}_c & \boldsymbol{\mu}_c \end{bmatrix}, \qquad \hat{\boldsymbol{z}} := \begin{bmatrix} \boldsymbol{z} \\ 1 \end{bmatrix},$$

as we have $\hat{\boldsymbol{\Lambda}}_c \hat{\boldsymbol{z}} = \boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c$. By making use of the symmetry of $\boldsymbol{D}_c^{-1}$, the cyclic property of the trace, and the linearity of $\mathbb{E}_{q_{n,c}}[\cdot]$, the expectation in Eq. (B.8) reformulates to

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

Inserting the aforementioned results into the free energy yields

$$\begin{aligned}
\mathcal{F}(\boldsymbol{x}_{1:N}; \boldsymbol{\mathcal{K}}, \tilde{\boldsymbol{\Theta}}, \boldsymbol{\Theta}) = \sum_{n,c} q_n(c;\,\tilde{\boldsymbol{\Theta}}) \Big(&\log \pi_c - \tfrac{D}{2} \log(2\pi) + \tfrac{1}{2} \log |\boldsymbol{D}_c^{-1}| - \tfrac{1}{2} \boldsymbol{x}_n^\top \boldsymbol{D}_c^{-1} \boldsymbol{x}_n \\
&+ \boldsymbol{x}_n^\top \boldsymbol{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}}\left[\hat{\boldsymbol{z}}\right] - \tfrac{1}{2} \operatorname{tr}\left(\hat{\boldsymbol{\Lambda}}_c^\top \boldsymbol{D}_c^{-1} \hat{\boldsymbol{\Lambda}}_c \mathbb{E}_{q_{n,c}}\left[\hat{\boldsymbol{z}} \hat{\boldsymbol{z}}^\top\right]\right) \Big) + \mathcal{H}\left[q\right], \qquad \text{(B.9)}
\end{aligned}$$

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

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

Based on this expression of the free energy, we derive the update equations by equating the partial derivatives of Eq. (B.9) 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

$$\pi_c = \frac{N_c}{N} \quad \text{with} \quad N_c = \sum_n q_n(c;\, \tilde{\boldsymbol{\Theta}}). \tag{B.10}$$

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

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

resulting in

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

where we introduced the following two matrices:

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

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

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

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

It follows that

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

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

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

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

$$\begin{aligned}\sigma_{cd}^2 &= \frac{1}{N_c}\,\mathrm{diag}\left(\sum_n q_n(c;\,\tilde{\boldsymbol{\Theta}})\boldsymbol{x}_n\boldsymbol{x}_n^\top - \boldsymbol{Y}_c\hat{\boldsymbol{\Lambda}}_c^\top\right)_d \\ &= \frac{1}{N_c}\left(\sum_n q_n(c;\,\tilde{\boldsymbol{\Theta}})\;x_{nd}^2 - \sum_h\left(\boldsymbol{Y}_c\odot\hat{\boldsymbol{\Lambda}}_c\right)_{d,h}\right),\end{aligned} \tag{B.11}$$

where all non-diagonal elements are set to zero and $\hat{\boldsymbol{\Lambda}}_c$ are the updated parameters given by Eq. (B.11). Here, $\mathrm{diag}(\cdot)$ denotes a vector constructed by the main diagonal of a given matrix and $\odot$ denotes element-wise multiplication.

Given that $q_n(c;\,\tilde{\boldsymbol{\Theta}})$ is zero for $c \notin \mathcal{K}^{(n)}$, the computational efficiency of the above equations can be improved by summing solely over non-zero $q_n(c;\,\tilde{\boldsymbol{\Theta}})$, as elaborated in Sec. 3.2.1 in the main text.

B.1.3 Proof of Proposition 1

Let $\boldsymbol{\mathcal{K}}$ 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 $\tilde{\boldsymbol{\mathcal{K}}}$. Now, we obtain:

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

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 $p(\tilde{c},\boldsymbol{x}_n\,|\,\boldsymbol{\Theta}) < p(c,\boldsymbol{x}_n\,|\,\boldsymbol{\Theta})$. For $p(\tilde{c},\boldsymbol{x}_n\,|\,\boldsymbol{\Theta}) = p(c,\boldsymbol{x}_n\,|\,\boldsymbol{\Theta})$ the free energy trivially remains unchanged. □

B.1.4 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 (Eq. (3.12) in the main text). Afterwards, we compare $D_{c\tilde{c}}$ to a previously suggested distance estimate in Appendix B.1.4.1.

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 Sec. 3.3 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 $\boldsymbol{x}_n$. 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 Eq. (3.12) 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 (Eq. (3.10) in the main text) using M samples of $p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta})$, which becomes exact for $M \to \infty$:

$$D_{\mathrm{KL}}\left[p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) \;||\; p(\boldsymbol{x} \,|\, \tilde{c}, \boldsymbol{\Theta})\right] \approx \frac{1}{M} \sum_{m=1}^{M} \log \frac{p(\boldsymbol{x}_m \,|\, c, \boldsymbol{\Theta})}{p(\boldsymbol{x}_m \,|\, \tilde{c}, \boldsymbol{\Theta})}\,, \quad \text{where} \quad \boldsymbol{x}_m \sim p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}). \tag{B.12}$$

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 Eq. (3.11) in the main text will contain data points close to those that would be generated by $p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta})$. Therefore, we can further approximate Eq. (B.12) by:

$$D_{\mathrm{KL}}\left[p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) \;||\; p(\boldsymbol{x} \,|\, \tilde{c}, \boldsymbol{\Theta})\right] \approx \frac{1}{|\mathcal{I}_c|} \sum_{n \in \mathcal{I}_c} \log \frac{p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})}{p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})}\,. \tag{B.13}$$

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 $p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})$ that are not performed during the partial E-step. A disadvantage is that the approximation quality may decrease as potentially fewer data points $\boldsymbol{x}_n$ 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:

$$\begin{aligned} D_{\mathrm{KL}}\left[p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) \;||\; p(\boldsymbol{x} \,|\, \tilde{c}, \boldsymbol{\Theta})\right] &\approx \frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log \frac{p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})}{p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})} \delta(\tilde{c} \in \mathcal{S}^{(n)}) = D_{c\tilde{c}}\,, \\ &\text{where} \quad N_{c\tilde{c}} = \sum_{n \in \mathcal{I}_c} \delta(\tilde{c} \in \mathcal{S}^{(n)}) \approx \sum_{n \in \mathcal{I}_c} 1 = |\mathcal{I}_c|. \end{aligned} \tag{B.14}$$

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 $p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})$ being almost equal to $p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})$, the summands in Eq. (B.14) 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 divert 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 Eq. (3.13) in the main text.

To summarize, $D_{c\tilde{c}}$ serves as an approximation for the KL-divergence between $p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})$ and $p(\boldsymbol{x}_n \,|\, \tilde{c}, \boldsymbol{\Theta})$, 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. Furthermore, $D_{c\tilde{c}}$ preserves sufficient information about the KL-divergence and, most importantly, it is efficiently computable.

B.1.4.1 Comparison to Variational E-Steps of Previous GMM Optimizations

Previous work using variational E-steps (Exarchakis *et al.*, 2022; Hirschberger *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

$$d^2_{c\tilde{c}} := -\frac{1}{N_{c\tilde{c}}} \sum_{n \in \mathcal{I}_c} \log p(\tilde{c}, \boldsymbol{x}_n \,|\, \boldsymbol{\Theta}) \delta(\tilde{c} \in \mathcal{S}^{(n)}), \tag{B.15}$$

where $N_{c\tilde{c}}$ is defined as in Eq. (3.12) in the main text. The estimate $d^2_{c\tilde{c}}$ was then used as a ranking to define sets g_c analogously to Eq. (3.13) in the main text.

For isotropic components with equal variances and equal mixing proportions per component, it was argued that $d^2_{c\tilde{c}}$ will finally correspond to the Euclidean component-to-component distance (and $d^2_{c\tilde{c}}$ was originally defined for this isotropic case). More concretely, it was shown (Appendix E of Hirschberger *et al.*, 2022) that $d^2_{c\tilde{c}}$ 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, $d^2_{c\tilde{c}}$ was also related to KL-divergences.

From the perspective of the here used approach in Eq. (3.12) in the main text, however, Eq. (B.15) can be considered as ignoring $p(\boldsymbol{x}_n \,|\, c, \boldsymbol{\Theta})$ in the KL-divergence (Eq. (3.10) in the main text). A further difference is a different consideration of the mixing proportions. Estimate Eq. (B.15) has been shown to perform well for isotropic and diagonal components (Hirschberger *et al.*, 2022), for which it was originally derived. Here, we 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 Eq. (3.12) in the main

text is improving learning remains to be numerically investigated, which we provide in Appendix B.2.2.1.

B.1.5 Algorithmic Complexity of the Variational E-step

To determine the algorithmic complexity of Alg. 1 in the main text, we analyze each of its four blocks, assuming $N > C$. Algorithm 4 outlines the steps in Alg. 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, whereas each joint computation has a complexity of $\mathcal{O}(DH)$ (see Appendix B.1.1). 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 dataset 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 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.

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 Alg. 4 is $\mathcal{O}(NSDH + NC' + NS + NC') = \mathcal{O}(NSDH)$.

Algorithm 4 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 Sec. 3.1 in the main text. Second, computing a joint defined by the MFA model requires $\mathcal{O}(HD)$ 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 Sec. 3.2.3 in the main text, Appendix B.1.4.1 and Appendix B.2.2.1).

Algorithm 4: Complexity of the Variational E-step

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

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

B.2 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 datasets in Appendix B.2.1, results of additional control experiments in Appendix B.2.2 and supplementary information on the quality analysis in Appendix B.2.3.

B.2.1 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 datasets.

B.2.1.1 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 Secs. 3.2.1 to 3.2.4 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$^{\text{Eucl.}}$, v-ISO and v-ISO$^{\text{Eucl.}}$ used in Appendix B.2.2 are also provided in this source code.

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[9].

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

$$\mathcal{P}_c = \{n \mid c = c_n\} \quad \text{with} \quad c_n = \underset{c'}{\operatorname{argmin}} \, ||\boldsymbol{x}_n - \boldsymbol{\mu}_{c'}||^2 \tag{B.16}$$

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 $\boldsymbol{D}_c$ for the corresponding cluster c. The priors π_c are determined based on the relative subset sizes, i.e.,

$$\pi_c = \frac{|\mathcal{P}_c|}{N}, \tag{B.17}$$

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 Eq. (3.15) in the main text. The original function does a fixed number of iterations. Unlike em-MFA and v-MFA, where the free energy is computed for the entire dataset during each E-step, *torch-mfa* updates parameters after each batch, making full dataset evaluation impractical. Instead, we computed the free energy on a randomly selected subset of the dataset after each epoch and used this for the convergence criterion in Eq. (3.15) in the main text. The subset size was set to 10× the batch size, and the computational time for this evaluation was excluded from the time measurement.

[8] `https://github.com/variational-sublinear-clustering/vamm`
[9] `https://github.com/variational-sublinear-clustering/emmi`

AFK-MC^2: We provide an efficient implementation of the AFK-MC^2 seeding method (Bachem *et al.*, 2016a) integrated into v-MFA. For all experiments, the Markov chain length was set to 10.

B.2.1.2 Hardware

Table B.1 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. An exception to this is the training of v-MFA on YFCC100M in Sec. 3.3.3 in the main text, where we executed only one instance at a time due to memory constraints, 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 $\approx$ 1 h vs. CPU (16 cores) $>$ 10 h for a single epoch).

Algorithm	Hardware		
	Device	Model name	Cores used
v-MFA	CPU	AMD Genoa EPYC 9554	16*
v-MFA on YFCC100M	CPU	AMD Genoa EPYC 9554	64
em-MFA	CPU	AMD Genoa EPYC 9554	16*
k-means+*FA*	CPU	AMD Genoa EPYC 9554	16*
torch-mfa	GPU	NVIDIA H100 94GB SXM	all

Tab B.1: 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.

B.2.1.3 Datasets and Preprocessing

In the following, we provide additional information on the datasets used in the numerical experiments. The main properties of the datasets are listed in Tab. B.2. Unless otherwise stated, we use the official *train* and *test* splits, and the images are used as provide by there sources without preprocessing. Note that the label information was not used in any of the experiments.

CIFAR-10: The CIFAR-10 dataset (Krizhevsky, 2009) consists 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 dataset (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 dataset (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 dataset (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 dataset 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 dataset (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 dataset) and used the remaining 94 197 533 samples for training.

Tab B.2: High-dimensional datasets used in our experiments. The data dimension is given in pixel width $\times$ pixel height $\times$ color channels.

dataset	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

B.2.2 Numerical Control Experiments

The following subsections report the results of additional control experiments, offering supporting or complementary insights to those discussed in Sec. 3.3 in the main text.

B.2.2.1 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 Eq. (3.12) in the main text, with a variant of v-MFA (referred to as v-MFA$^{\text{Eucl.}}$) that uses the definition of g_c introduced in Hirschberger *et al.* (2022), based on $d^2_{c\tilde{c}}$ in Eq. (B.15). In addition to the MFA model, we also consider isotropic GMMs (compare Eq. (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 $d^2_{c\tilde{c}}$ (referred to as v-ISO$^{\text{Eucl.}}$). v-ISO$^{\text{Eucl.}}$ is the algorithm used in previous work (Hirschberger *et al.*, 2022).

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

The models are initialized identically and using the same settings as described in Sec. 3.3.2 and Fig. 3.5 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$^{\text{Eucl.}}$ (or v-ISO$^{\text{Eucl.}}$) for the same number of iterations.

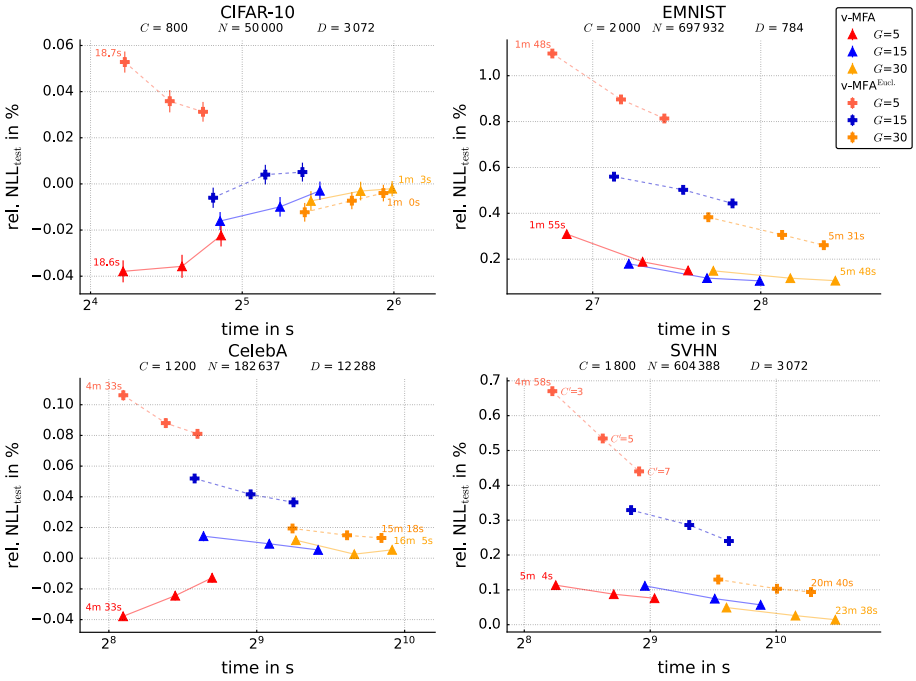

Fig B.1: Effectiveness and efficiency of v-MFA and v-MFA$^{\text{Eucl.}}$ in terms of NLL_{test} (relative to em-MFA) and the runtime. Triangles refer to v-MFA, while pluses denote v-MFA$^{\text{Eucl.}}$. Each of the four subfigures corresponds to experiments on one benchmark dataset. The settings are similar to those in Fig. 3.5 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 comparison results for the MFA models and for the isotropic GMMs are presented in Fig. B.1 and Fig. B.2, respectively. In the case of MFAs, the here derived v-MFA algorithm consistently performs better (or comparable within the error bars) in terms of NLL_{test} values when compared to v-MFA$^{\text{Eucl.}}$ (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$^{\text{Eucl.}}$ albeit at a slightly higher computational cost, due to smaller intersections in the search spaces. Notably, v-ISO$^{\text{Eucl.}}$ only differs slightly from v-ISO in terms of NLL_{test} values for most of the datasets and settings of C' and G (see Fig. B.2), which confirms the suitability $d^2_{c\tilde{c}}$ of Eq. (B.15) 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$^{\text{Eucl.}}$ highlight the importance of the more general definition of g_c using $D_{c\tilde{c}}$ given by Eq. (3.12) in the main text.

B.2.2.2 Comparison to other Implementations

In experiments in Sec. 3.3.1 and Sec. 3.3.2 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). To ensure a fair comparison, the primary motivation for developing v-MFA

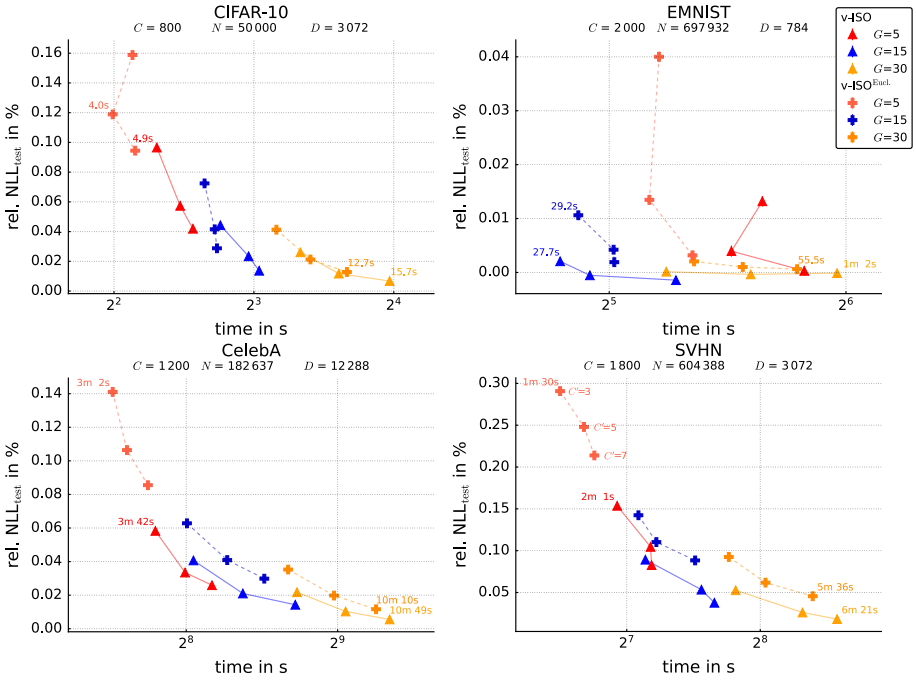

Fig B.2: Effectiveness and efficiency of v-ISO and v-ISO$^{\text{Eucl.}}$ in terms of NLL_{test} (relative to em-MFA) and the runtime. Triangles refer to v-ISO, while pluses denote v-ISO$^{\text{Eucl.}}$. Each of the four subfigures corresponds to experiments on one benchmark dataset. The settings are similar to those in Fig. 3.5 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).

and em-MFA was to optimize execution speed. To investigate their efficiency against existing algorithms in the literature, we compared them with an already published implementation of MFA optimization (Richardson and Weiss, 2018) which is based on stochastic gradient descent (SGD) (but is not using variational acceleration). Concretely, we used the *torch-mfa* algorithm which is based on the PyTorch framework and uses its integrated optimizers for SGD. *torch-mfa* represents a further advancement of the TensorFlow implementation employed to produce the previously reported results (Richardson and Weiss, 2018) (for further details, see Appendix B.2.1.1). To the best of the authors' knowledge, the work by Richardson and Weiss (2018) documented the largest MFA models prior to our study. However, at the scales we here address, 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.

To assess the performance of *torch-mfa* against v-MFA and em-MFA, we adopted the configurations used in an experiment on the CelebA dataset (see Richardson and Weiss, 2018). The number of components and the hyperplane dimension are set to $C = 1000$ and $H = 10$ for all algorithms. We ran *torch-mfa* on a GPU with the configurations specified

in Appendix B.2.1. This is in contrast to the CPU configurations for the other algorithm, as discussed in Appendix B.2.1.2. 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 dataset, without hierarchical training. In addition to *torch-mfa*, v-MFA and em-MFA, we used the k-means+*FA* algorithm in for our experimental comparison. The results are shown in Tab. B.3.

Tab B.3: Comparison of em-MFA, v-MFA, k-means+*FA* and *torch-mfa* on CelebA with $C = 1000$. To ensure consistency with the other methods, we define an iteration for *torch-mfa* as completed when the model has processed the entire dataset, 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}, runtime (d=days, h=hours, m=minutes, s=seconds) and the number of joint or distance evaluations are marked bold.

	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	$\mathbf{57565 \pm 2}$	**(-0.01 $\pm$ 0.01) %**	**9m 2s $\pm$ 11.6s**	$\mathbf{(2.47 \pm 0.05) \cdot 10^8}$	35.0
k-means+*FA*	58373 ± 2	(1.39 ± 0.00) %	12m 12s $\pm$ 2.8s	$(1.83 \pm 0.01) \cdot 10^{10}$	106.7
torch-mfa	57868 ± 13	(0.52 ± 0.02) %	1d 14h $\pm$ 1h 35m	$(6.8 \pm 0.3) \cdot 10^9$	38.5

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. In terms of computational efficiency, 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 dataset), *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 the lowest value with 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.

B.2.2.3 Control Experiments for the v-MFA Scalability Analysis

The results in Fig. 3.4 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 Sec. 3.3.1 in the main text, now incorporating the different optimization qualities as measured by the NLL on the test datasets. Concretely, we stopped the em-MFA algorithm once it achieved the same $\mathrm{NLL}_{\mathrm{test}}$ as was achieved after a full run of the v-MFA algorithm.

As we only evaluate $\mathrm{NLL}_{\mathrm{test}}$ 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 $\mathrm{NLL}_{\mathrm{test}}$ after each iteration during training. When the $\mathrm{NLL}_{\mathrm{test}}$ 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 em-MFA† in Fig. B.3, i.e., em-MFA† 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 Fig. B.3 are taken from Fig. 3.4 in the main text. For two of the four datasets, em-MFA† achieved a smaller scaling exponent than em-MFA. The number of iterations required to obtain the same $\mathrm{NLL}_{\mathrm{test}}$ as v-MFA decreases slightly with increasingly large values of C. For both CIFAR-10 and CelebA, the scaling exponents of em-MFA† and em-MFA are approximately the same. On CIFAR-10, the $\mathrm{NLL}_{\mathrm{test}}$ values of v-MFA are lower than those of em-MFA for sufficiently large values of C, as also observed in Fig. 3.5 in the main text. Consequently, the algorithm requires more iterations than previously to reach the desired quality.

Overall, the scaling exponent of v-MFA remains significantly lower than that of em-MFA† across all four datasets. 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.2.3 Additional Information on the Quality Analysis

Additional information and results about the experiments in Sec. 3.3.2 in the main text are provided in Tab. B.4. The table reports the relative $\mathrm{NLL}_{\mathrm{test}}$, 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).

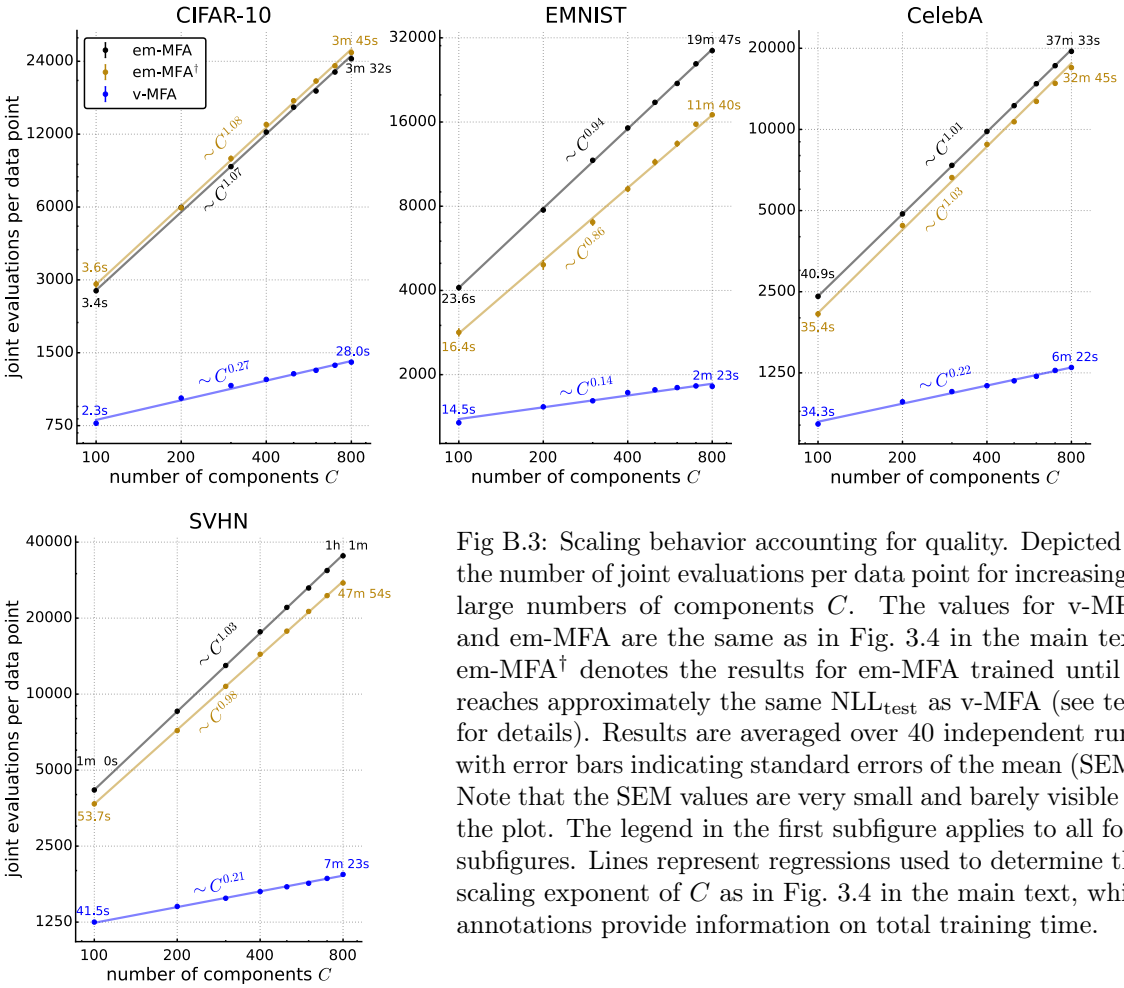

Fig B.3: Scaling behavior accounting for quality. Depicted is the number of joint evaluations per data point for increasingly large numbers of components C. The values for v-MFA and em-MFA are the same as in Fig. 3.4 in the main text. em-MFA† denotes the results for em-MFA trained until it reaches approximately the same $\mathrm{NLL}_{\mathrm{test}}$ 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 Fig. 3.4 in the main text, while annotations provide information on total training time.

Tab B.4: Additional information and results for the experiments in Sec. 3.3.2 in the main text. Standard errors of the mean (SEM) for relative NLL_{test} were below 0.01%.

Dataset	**Algorithm**				Rel. speed-up		#Iterations	
#Components	Name	C'	G	rel. NLL_{test}	Time	Joints/Dist.	*warm-up*	total
CIFAR-10	em-MFA	-	-	0.0%	1.0×	1.0×	-	31.0
$C = 800$	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	em-MFA	-	-	0.0%	1.0×	1.0×	-	29.0
$C = 2000$	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	em-MFA	-	-	0.0%	1.0×	1.0×	-	24.0
$C = 1200$	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	em-MFA	-	-	0.0%	1.0×	1.0×	-	42.0
$C = 1800$	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

Acknowledgments. We thank Jan Warnken, Roy Friedman and Yair Weiss for valuable discussions and feedback.

During the preparation of this work the authors partly used ChatGPT[10] and DeepL[11] 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.

[10] `https://chatgpt.com/`
[11] `https://www.deepl.com/`

Appendix C

Supplementary Information on Variational Graph-Based Semi-Supervised Learning

C.1 Details on Numerical Experiments

Below, we provide additional details on the numerical experiments. This includes supplementary information on the datasets in Appendix C.1.1 and preprocessing procedures in Appendix C.1.2, along with further details regarding the implementations and hyperparameters of the algorithms in Appendix C.1.3, the algorithms' initialization in Appendix C.1.4, the training of MFAs in Appendix C.1.5, and the hardware used in Appendix C.1.6.

C.1.1 Datasets

In the following, we describe the datasets used in our numerical experiments. When applicable, *train* and *test* splits were merged to increase the amount of data.

USPS: The USPS dataset (Hull, 1994) consists of 9298 grayscale images of handwritten digits, sized at 16×16 pixels. The dataset includes 10 label classes corresponding to the digits $0-9$. We use a version of USPS where the pixel values are between 0 and 255.

Letter: The Letter dataset (Slate, 1991) comprises 20 000 samples, each represented by 16 numerical attributes describing a letter. The 26 label classes correspond to the capital letters of the English alphabet.

CIFAR-10: The CIFAR-10 dataset (Krizhevsky, 2009) consists of 60 000 natural color images, each measuring 32×32 pixels. The dataset is divided into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

MNIST: The MNIST dataset (Lecun *et al.*, 1998) contains 70 000 grayscale images of handwritten digits, sized at 28×28 pixels. It includes 10 label classes corresponding to the digits $0-9$.

EMNIST: The EMNIST dataset (Cohen *et al.*, 2017) is an extension of MNIST, comprising grayscale images of handwritten characters sized 28×28. The images are split into different

groups. We used the *digits* split, which includes 280 000 images of handwritten digits with labels $0 - 9$.

SUSY: The SUSY dataset (Baldi *et al.*, 2014) contains 5 000 000 particle physics collision events from Monte Carlo simulations. Each event is described by 18 features: the first 8 features are directly measured from the state of the particles, while the last 10 features are derived non-linear transformations of the measured features. The binary labels indicate whereas an event is a background or supersymmetric particle event.

MNIST8M: The MNIST8M dataset (Loosli *et al.*, 2007) comprises 8 100 000 grayscale images of handwritten digits, each of size 28×28, with labels $0 - 9$. MNIST8M is derived from the MNIST dataset through data augmentation, applying pseudo-random deformations and translations. We obtained it using the tool InfiMNIST[12].

C.1.2 Preprocessing

We adopted preprocessing pipelines similar to those described in Zhang *et al.* (2023). For the low-dimensional datasets, Letter and SUSY, no preprocessing was performed. For USPS, MNIST, EMNIST, and MNIST8M, dimensionality reduction was applied using Principal Component Analysis (PCA) to project the data onto 30 dimensions. We used the PCA implementation from scikit-learn (version 1.5.2), with full Singular Value Decomposition (SVD) and all other parameters set to their default values. An exception is MNIST8M, where we used the '`covariance_eigh`' solver instead of full SVD for PCA, due to the large number of data points.

For CIFAR-10, the images were (cf. Zhang *et al.*, 2023) first processed by a ResNet-34 model, with weights pre-trained on ImageNet (He *et al.*, 2016). We used PyTorch to obtain the ResNet-34[13]. Feature vectors were extracted before the final fully connected layer (FC layer), yielding 512-dimensional representations. These feature vectors were further reduced to 30 dimensions using PCA.

C.1.3 Details on Implementations and Hyperparameters of the Algorithms

In this work, we reimplemented all algorithms for which the original implementations were either in MATLAB or unavailable. Concretely, we translated the official MATLAB implementations of ARG (Liu *et al.*, 2010), EAGR (Wang *et al.*, 2016), f-FME and r-FME (Qiu *et al.*, 2019) into Python, ensuring our implementations closely follow the original codes. By comparing the inferred labels from our implementations with those from the MATLAB versions (label by label), we verified that both produced the same outputs. Additionally, we observed faster runtimes for our implementations. Regarding MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$ (Chi *et al.*, 2010) and regarding DDGL (Zhang *et al.*, 2023) there are (to the best of our knowledge) no official implementations publicly available. Therefore, we implemented these algorithms based on the methodologies described in Chi *et al.* (2010) and Zhang *et al.* (2023), respectively.

For all algorithms (the here introduced ones and the benchmark algorithms) we followed a fixed comparison protocol. Concretely, we used for a given algorithm one set of hyperparameters consistently across all experiments, i.e., for none of the algorithms any

[12] `https://leon.bottou.org/projects/infimnist`
[13] `https://pytorch.org/vision/main/models/generated/torchvision.models.resnet34.html`

hyperparameter fine-tuning was done on individual datasets or settings. The selection of the hyperparameters for a given algorithm was carried out as follows: whenever possible, we adopt values recommended in the original work. For algorithms sharing common parameters (e.g., C or γ), the same values are used. For the remaining hyperparameters, preliminary experiments were conducted to find values that exhibit robust and good performance across all datasets.

In the following, we will provide implementation details and details of the hyperparameters and their values for each algorithm. For the benchmarking algorithms we remark that below descriptions do not replace the detailed descriptions of the original works (i.e., we refer to the corresponding papers for more details). The used source code of all algorithms is available here [14].

We begin by explaining the algorithms which are used for distribution learning by several GSSL methods. Subsequently, we provide details for each GSSL algorithm.

***k*-means:** For k-means clustering (Lloyd's algorithm) to identify anchor/landmark nodes, we use the implementation from scikit-learn (version 1.5.2) with default hyperparameters unless otherwise specified.

The number of components is set to $C = 1000$ across all experiments. The k-means algorithm is trained until the default convergence criterion is met, with a tolerance of $\varepsilon = 10^{-4}$ for all datasets except SUSY. For SUSY, where $\varepsilon = 10^{-4}$ results in excessive iterations, we use $\varepsilon = 10^{-3}$.

Conventional GMM Optimization: GMMs with diagonal and shared covariance for DDGL, MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$ as well as MFAs for MFA-GL, are optimized using the conventional EM algorithm (using full posteriors). We utilize the implementation used in Salwig *et al.* (2025)[15].

The number of components is set to $C = 1000$ across all experiments. We use the same convergence criterion as in Hirschberger *et al.* (2022) and Salwig *et al.* (2025). The threshold for convergence is set in analogy to k-means above, i.e., we use $\varepsilon = 10^{-4}$ for all datasets except of SUSY, where we use $\varepsilon = 10^{-3}$.

Variational GMM Optimization: For v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL, the model parameters are optimized using the variational EM algorithm proposed in Salwig *et al.* (2025)[16]. The number of components is set to $C = 1000$ across all experiments, except for the hyperparameter analysis in Appendix C.2.4. As with conventional EM training, we use the same convergence criterion as in Hirschberger *et al.* (2022) and Salwig *et al.* (2025). The threshold for convergence is set in analogy to conventional EM optimization and k-means above, i.e., we use $\varepsilon = 10^{-4}$ for all datasets except of SUSY, where we use $\varepsilon = 10^{-3}$. The hyperparameters of the variational approximations were set to $C' = 3$ and $G = 15$ across all experiments, which was previously observed to result in a good and robust performance (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Salwig *et al.*, 2025). $S = C'G + 1$ defines the size of the search spaces (cf. Sec. 4.2.3 and Salwig *et al.*, 2025).

LAE and FLAE: The Local Anchor Embedding (LAE) and Fast Local Anchor Embedding (FLAE) algorithms are reimplemented in C++ using the open-source library Eigen.

[14] `https://github.com/salwig/gssl`
[15] `https://github.com/variational-sublinear-clustering/emmi`
[16] `https://github.com/variational-sublinear-clustering/vamm`

The hyperparameters of LAE and FLAE were consistently set to the default values specified in their respective publications across all experiments. Concretely, for LAE: the number of the closest anchors was set to $s = 3$ and the number of iterations was set to $cn = 10$; and for FLAE: the number of the closest anchors was set to $s = 3$ and the regularization hyperparameter β was set to 1.0 (note that this hyperparameter is different from the inverse temperature β used for v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL).

AGR: We reimplemented AGR based on the methodology described in Liu *et al.* (2010) and the source code provided on GitHub[17]. The AGR algorithm uses the k-means algorithm and the LAE algorithm. We implemented the remaining part of the algorithm in Python using the libraries NumPy and SciPy.

Apart from the hyperparameters associated with k-means and LAE, AGR includes the fitting/smoothness trade-off hyperparameter, which was set to $\gamma = 1.0$ to ensure consistency with the other algorithms.

EAGR: We reimplemented EAGR based on the methodology described in Wang *et al.* (2016) and the source code provided on GitHub[18]. The EAGR algorithm uses the k-means algorithm and the FLAE algorithm. We implemented the remaining part of the algorithm in Python using the libraries NumPy and SciPy.

Apart from the hyperparameters associated with k-means and FLAE, EAGR includes the fitting/smoothness trade-off hyperparameter, which was set to $\gamma = 1.0$ to ensure consistency with the other algorithms.

f-FME and r-FME: We reimplemented f-FME and r-FME based on the methodology described in Qiu *et al.* (2019) and the source code provided on GitHub[19]. Both f-FME and r-FME are based on the k-means algorithm. The f-FME algorithm employs the LAE algorithm, whereas r-FME incorporates the FLAE algorithm. We implemented the remaining parts of the algorithms in Python using the libraries NumPy and SciPy.

The hyperparameter γ was set to $\gamma = 1.0$ to ensure consistency with the other algorithms. The additional hyperparameters inherent to f-FME and r-FME were consistently chosen across all experiments as follows: The first labeled and the rest unlabeled diagonal elements of the diagonal matrix U (for more details, we refer to Qiu *et al.*, 2019) were set to $u_l = 10^9$ and $u_u = 0.0$, respectively, as specified in the original work. For the hyperparameter μ, no default value was reported. In our experiments, we used the value $\mu = 10^{-3}$, which is commonly reported in the experiments of Qiu *et al.* (2019) and showed good performance on preliminary tests.

MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$: We implemented MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$ following the methodology described in Chi *et al.* (2010). Both MiMoLaP algorithms use the conventional EM algorithm. We implemented the remaining parts of the algorithms in Python using the library PyTorch. To make MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$ applicable on our benchmarks, we restricted the covariances to be diagonal and shared among the components. Without constraints, these algorithms would have (per iteration) the $\mathcal{O}(NCD^2)$ complexity of full GMMs. Therefore, they would have runtimes even larger than MFA-GL. The here applied constraint also simplifies the computation of the probability product kernel (used

[17] `https://github.com/ColumbiaDVMM/Anchor-Graph`
[18] `https://github.com/fuweijie/EAGR`
[19] `https://github.com/kyuusaku/fFMEsemi`

in MiMoLaP$^{\mathrm{LG}}$ and MiMoLaP$^{\mathrm{H}}$), as using full covariances would substantially increase the computational complexity also in that stage of the algorithm.

The hyperparameter of the probability product kernel was set to $\rho = 0.5$ (as specified in Chi *et al.*, 2010). The fitting/smoothness trade-off hyperparameter used by MiMoLaP$^{\mathrm{LG}}$ was set to $\gamma = 1.0$ to ensure consistency with the other algorithms. MiMoLaP$^{\mathrm{H}}$ does not include the hyperparameter γ.

DDGL: We implemented the DDGL algorithm following the methodology described in Zhang *et al.* (2023). The DDGL algorithm uses the conventional EM algorithm. We implemented the remaining parts of the algorithm in Python using the library PyTorch. We also provide a version of DDGL with a sparse graph learning and labeling stage (for more details, see Appendix C.3.1). For this version, we utilize the Python libraries NumPy and SciPy.

Apart from the hyperparameters associated with conventional EM, DDGL includes the fitting/smoothness trade-off hyperparameter, which was set to $\gamma = 1.0$ to ensure consistency with the other algorithms.

MFA-GL: The MFA-GL algorithm is developed as part of this work. It is similar to the v-MFA-GL algorithm, with the following differences: It uses the conventional EM algorithm. The transformation matrix $\boldsymbol{Z}$ in Eq. (4.4) in the main text is constructed using the full posterior distributions instead of truncated posteriors, and, furthermore, no hot posteriors (as defined in Eq. (4.14) in the main text) are employed. Additionally, we directly compute the soft class matrix $\boldsymbol{A}$ by Eq. (4.9) from the main text instead of utilizing the conjugate gradient method as in Alg. 3. We implemented the MFA-GL algorithm in Python using the library PyTorch.

As for DDGL, we provide a version of MFA-GL with a sparse graph learning and labeling stage (for more details, see Appendix C.3.1). For this version, we utilize the Python libraries NumPy and SciPy.

Apart from the hyperparameters associated with conventional EM, MFA-GL includes the hyperplane dimensionality H and the fitting/smoothness trade-off hyperparameter γ. The hyperplane dimensionality was set to $H = 5$, and the hyperparameter γ was set to $\gamma = 1.0$ to ensure consistency with the other algorithms.

v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL: The v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL algorithms are developed as part of this work. They are based on the variational EM algorithm. We implemented the core functionality of these algorithms in C++ using the libraries Eigen and Boost.

Apart from the hyperparameters associated with variational EM, v-GMM$^{\mathrm{d}}$-GL as well as v-MFA-GL include the fitting/smoothness trade-off hyperparameter γ and the inverse temperature β. For v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL we fixed β to the same value to construct the transformation matrix $\boldsymbol{Z}$ (note that we do not apply the hot posteriors *during* distribution learning, i.e., we use Eq. (4.1) from the main text for the variational EM algorithm). For v-GMM$^{\mathrm{d}}$-GL we set $\beta = 1/\sqrt{D}$, and for v-MFA-GL we set $\beta = 1/D$. Additionally, the v-MFA-GL algorithm includes the hyperplane dimensionality, which we set to $H = 5$ (see Appendix C.2.4).

C.1.4 Initialization

We employ AFK-MC^2 (Bachem *et al.*, 2016a) to efficiently sample data points as initial seeds for the cluster centers for k-means or component means for GMMs. For all experiments, the Markov chain length for AFK-MC^2 was set to $m = 10$. The mixing proportions π_c are uniformly initialized with $1/C$. To initialize the diagonal variances $\boldsymbol{S}$ for the diagonal GMMs, we compute the variance along each dimension across all data points as follows:

$$\boldsymbol{S}_{\text{init}} = \text{diag}(\sigma_1^2, \cdots, \sigma_D^2) \text{ and } \sigma_d^2 = \text{var}(x_{1:N,d}), \tag{C.1}$$

where $x_{n,d}$ is the d-th entry of $\boldsymbol{x}_n$. For the MFA model, we used uniform random numbers between 0 and 1 to set the initial values of the factor loading matrices $\boldsymbol{\Lambda}_c$, as in Salwig *et al.* (2025). The variational parameters for both v-GMM$^{\text{d}}$-GL and v-MFA-GL were initialized following the procedure detailed in Salwig *et al.* (2025). To ensure a fair comparison in the numerical experiments, all methods were initialized as consistently as possible. For instance, the same seeding data points were used to initialize the means or centers of all methods.

C.1.5 Training for MFA-GL and v-MFA-GL

During the training of the MFA models by MFA-GL and v-MFA-GL, the conventional EM or variational EM algorithms first use isotropic covariance matrices $\boldsymbol{\Sigma}_c = \sigma^2 \boldsymbol{I}$ and fixed mixing proportions $\pi_c = 1/C$ (until the free energy has converged, see Appendix C.1.3). After this initialization phase for means $\boldsymbol{\mu}_c$ and the diagonal elements of the covariance matrix, the full MFA model (again with all $\boldsymbol{\mu}_c$ but also with individual π_c and flexible covariance structure, i.e., $\boldsymbol{\Sigma}_c = \boldsymbol{\Lambda}_c \boldsymbol{\Lambda}_c^T + \boldsymbol{S}$) is trained until convergence. The initialization phase requires just a fraction of the overall training time, and we observed a better performance compared to a simultaneous optimization of all model parameters without this initialization phase.

C.1.6 Hardware

All experiments were conducted on AMD Genoa EPYC 9554 CPUs. To achieve high utilization, we executed four parallel instances of the GSSL methods, with each instance using 16 of the 64 available cores. Thus, four independent runs were executed at the same time.

C.2 Control Experiments

In this section, we present control experiments in addition to the results discussed in Sec. 4.3 in the main text. Experiments include consistency experiments for AGR, EAGR, f-FME and DDGL in Appendix C.2.1, comparison to further GSSL approaches in Appendix C.2.2, an ablation study for truncated and hot posteriors in Appendix C.2.3, and a hyperparameter analysis for v-GMM$^{\text{d}}$-GL and v-MFA-GL in Appendix C.2.4.

C.2.1 Consistency Experiments

To compare the proposed algorithms, v-GMM$^{\text{d}}$-GL, v-MFA-GL, and MFA-GL, with benchmark algorithms from the literature, we first verified that the algorithms that we reimplemented (see Appendix C.1.3) produced error rates consistent with the literature. We

compare in this section the error rates of AGR, EAGR, f-FME, and DDGL with those recently reported on the EMNIST dataset (cf. Zhang *et al.*, 2023). The results are depicted in Tab. C.1. For consistency with the results in Zhang *et al.* (2023), the values in Tab. C.1 report means and standard deviations computed based on our measurements. In contrast, we use in the main text and in Tab. C.5 medians and standard errors in the median (computed using the same measurements).

Tab C.1: Comparison of error rates in % obtained using AGR, EAGR, f-FME, and DDGL on the EMNIST dataset, along with values reported in the literature (Zhang *et al.*, 2023). Note that these values differ from those presented in Tab. C.5, as this comparison considers average values and standard deviations.

N_l		AGR	EAGR	f-FME	DDGL
50	Zhang *et al.* (2023)	10.78 ± 1.6	9.79 ± 1.63	10.15 ± 1.8	9.25 ± 2.07
	ours	10.55 ± 1.74	9.24 ± 1.61	9.98 ± 1.72	9.94 ± 2.26
100	Zhang *et al.* (2023)	8.99 ± 1.24	7.89 ± 0.88	8.48 ± 0.91	7.23 ± 0.87
	ours	8.84 ± 1.10	7.37 ± 0.88	8.27 ± 1.13	7.37 ± 1.22
150	Zhang *et al.* (2023)	7.67 ± 0.64	7.57 ± 0.9	7.95 ± 0.65	6.98 ± 0.87
	ours	8.01 ± 0.82	6.58 ± 0.57	7.57 ± 0.85	6.61 ± 0.79
200	Zhang *et al.* (2023)	7.3 ± 0.57	7.08 ± 0.63	7.79 ± 0.57	6.59 ± 0.58
	ours	7.54 ± 0.62	6.16 ± 0.42	7.22 ± 0.66	6.36 ± 0.66

The results confirm that the error rates produced by our (re)implementations are consistent with those reported in Zhang *et al.* (2023) (within the standard deviations).

C.2.2 Comparison to Further GSSL Approaches

We compare all the approaches benchmarked in Sec. 4.3 additionally to further GSSL algorithms. The additional algorithms were applied in Li *et al.* (2020) to the datasets Letter, EMNIST, and MNIST8M, and they include further distribution-based GSSL methods as well as graph neural networks. More concretely, we here additionally compare to: k-means Mixture-distribution based Graph Smoothing (KMGS), Gaussian Anchor Graph (GAG), Mixture-distribution based Graph Smoothing with Semi-supervised EM modeling (MGS) (all introduced in Li *et al.*, 2020), Harmonic Mixtures (HM; Zhu and Lafferty, 2005), Graph Convolutional Networks (GCN; Kipf and Welling, 2017) and Hierarchical Anchor Graph Regularization (HAGR; Wang *et al.*, 2017). In Li *et al.* (2020), different variants of MGS were proposed. Here, we compare to results of the version incorporating semi-supervised EM, as it consistently outperformed the other variants of MGS. For GCN, the subscript (2 or 3) indicate the depth (i.e., the number of layers) of the neural network. For HAGR, we use for comparison the results of the version with three hierarchical layers, which consistently outperformed the two-layer version in Li *et al.* (2020).

Most above methods are usually not as competitive as the ones we compare to in Sec. 4.3. The exceptions are MGS and HM, which can achieve lower error rates than DDGL in some settings. To the best of our knowledge, no source code is publicly available for MGS or HM, however, and reimplementations are not straight-forward. Therefore, we use the error rates as they were reported previously (Li *et al.*, 2020), but which are mostly for different numbers of labeled data points N_l than those we use in Sec. 4.3. For HM, we also remark that (due to a quadratic scaling with N for graph construction) the execution times stated

(cf. Li *et al.*, 2020) indicate significantly longer runtimes compared to those algorithms we compare in Sec. 4.3.1.

The comparison is depicted in Tab. C.2. Note that (like in Tab. C.1) we use averages in Tab. C.2 to be consistent with results reported in Li *et al.* (2020). For all approaches investigated in Sec. 4.3 we also report standard deviations, while for the approaches investigated in Li *et al.* (2020) only the averages were reported.

By considering Tab. C.2 and error rates of KMGS, GAG, MGS, HM, GCN and HAGR previously reported in Li *et al.* (2020), it can be observed that v-MFA-GL and MFA-GL achieve lower error rates while utilizing fewer labeled data points. For instance, on MNIST8M, v-MFA-GL attained an average error rate of 4.52 % using only 100 labeled data points, whereas the MGS approach achieved an average error rate of 4.89 % despite using ten times as many labeled data points ($N_l = 1000$). Similarly, for comparable numbers of labeled data points on Letter, or for the same values of N_l on EMNIST, v-GMM$^{\mathrm{d}}$-GL consistently outperformed KMGS, GAG, GCN_2, GCN_3, and HAGR. In several settings, v-GMM$^{\mathrm{d}}$-GL also achieved lower error rates than MGS and HM.

C.2.3 Ablation Study for Truncated and Hot Posteriors

In the experiments presented in Sec. 4.3.2, we observed that the v-MFA-GL algorithm consistently outperformed the MFA-GL algorithm (except on EMNIST and MNIST8M for few labels), despite both algorithms using (with MFA) the same data distribution model.

v-MFA-GL differs from MFA-GL by two aspects: (A) by using truncated posteriors and a corresponding transformation matrix $\boldsymbol{Z}$ for label propagation; and (B) by applying hot posteriors (Eq. (4.14)). Both aspects work hand-in-hand as hot posteriors keep the sparsity but even out the non-zero values. However, it may be interesting to investigate how relevant truncated posteriors are for performance, and how relevant their hot versions are in comparison. For such an investigation, we conducted control experiments on the USPS, Letter, CIFAR-10 and MNIST datasets (because these datasets showed the largest differences between v-MFA-GL and MFA-GL performance). For each dataset, we ran v-MFA-GL with hot posteriors (setting $\beta = 1/D$ as in Fig. 4.5) and without (setting $\beta = 1$). For comparison, we also ran MFA-GL and, additionally, a version of MFA-GL with hot posteriors (we also set $\beta = 1/D$, but now all posterior values are non-zero and are affected).

The results of our control experiments are depicted in Fig. C.1. First, we observe that for three out of the four experiments, performance of v-MFA-GL is slightly better or equal to MFA-GL when *no* hot posteriors are used (i.e., $\beta = 1$). This confirms earlier observations in studies (Exarchakis *et al.*, 2022; Hirschberger *et al.*, 2022; Salwig *et al.*, 2025) focused on distribution learning alone[20]. Only for USPS, v-MFA-GL shows a substantial performance increase, presumably due to MFA-GL being more susceptible to local optima for this data.

Second, for v-MFA-GL we observe that hot posteriors consistently improve performance compared to v-MFA-GL without hot posteriors. Notably, no fine-tuning of the inverse temperature was necessary. The improved performance may be due to hot posteriors

[20]The earlier studies observed speed-ups in training time for truncated posteriors without performance deficits or even with slight performance gains

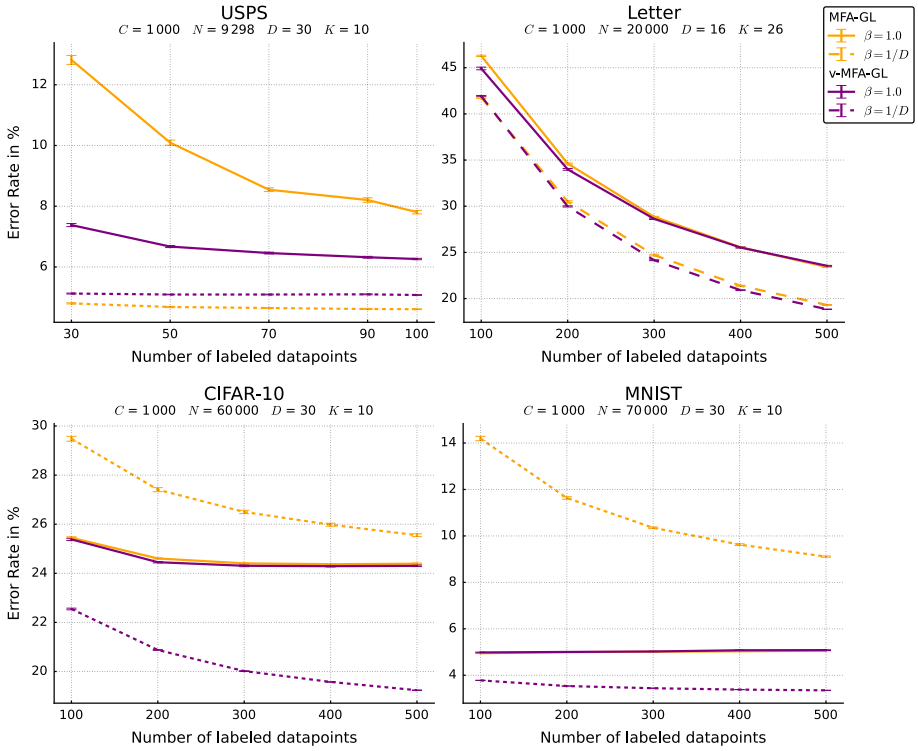

Fig C.1: Error rates in % for MFA-GL and v-MFA-GL, applied with $\beta = 1.0$ and $\beta = 1/D$. Each of the four subfigures corresponds to experiments on one benchmark dataset. Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling.

maintaining more valuable neighborhood information for label propagation that is suppressed by otherwise excessively small non-zero values. In contrast, hot posteriors do not necessarily improve performance if applied to the full posteriors of MFA-GL. They are consequently an inherent and consistent improvement of the variational approach here applied.

C.2.4 Hyperparameter Analysis

In this section, we examine the impact of several hyperparameters, including the number of components C, the hyperplane dimensionality H (only for v-MFA-GL), the fitting/smoothness trade-off hyperparameter γ and the inverse temperature β for the hot posteriors (Eq. (4.14) in the main text) for both v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL.

Recall that we used as default values $C = 1000$, $\gamma = 1$, $\beta = 1/\sqrt{D}$ for v-GMM$^{\mathrm{d}}$-GL, and the default values for v-MFA-GL were $C = 1000$, $\gamma = 1$, $\beta = 1/D$, and $H = 5$ (see Appendix C.1.3). These default values were used for all experiments in Sec. 4.3 in the main text.

Figure C.2 shows the change in the error rate if one of the hyperparameters is changed. It can be observed that the default values for v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL correspond to a good performance for all datasets. The algorithms are also robust across a large range of values for the hyperparameters. Notably, it can nevertheless be observed that performance would still further increase if hyperparameters were fine-tuned for each dataset. However, we did not use any such fine-tuning for v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL or for any of the other algorithms because such fine-tuning would result in a distorted comparison. This

is because ground-truth labels as they are used for the experiments of Fig. C.2 are not available in realistic SSL settings. We note that using in practice not available labels for hyperparameter tuning, e.g., by using large labeled validation sets, was explicitly criticized in the past (see, e.g., Forster *et al.*, 2018; Oliver *et al.*, 2018). Therefore, we use default values for all here compared algorithms across all datasets.

In the following we briefly discuss the dependence on each individual hyperparameter:

C: The number of components C can significantly influence error rates. In general, smaller values of C are advantageous for reducing computational complexity. However, larger C values enable the model to more accurately capture the underlying data distribution. Nonetheless, overly large C values can negatively affect error rates, particularly when the dataset size N is insufficient to support the increased model size. Consequently, C should be selected with consideration of the dataset size N.

H: The optimal value of the hyperplane dimensionality H varies depending on a dataset's inherent structure. For USPS, Letter and SUSY, $H = 2$ or 3 yields the lowest error rates, whereas for MNIST, EMNIST, and MNIST8M, the optimal value is approximately $H = 8$. On CIFAR-10, the impact of H on error rates is minimal. While the value $H = 5$, used in the main experiments in Sec. 4.3, is not the optimal choice for any specific dataset, it represents a choice with a good performance across all datasets. The observation that v-MFA-GL exhibits higher error rates on SUSY with both increasing H and C further suggests that the construction of the SUSY dataset is not consistent with the assumptions underlying the MFA model, as discussed in the main text.

γ: The framework demonstrates robustness to variations in the fitting/smoothness trade-off hyperparameter γ across several orders of magnitude. The value $\gamma = 1$, employed in Sec. 4.3, consistently achieves low error rates across all investigated datasets.

β: For the inverse temperature β, the results in Fig. C.2 indicate that the default values $\beta = 1/\sqrt{D}$ for v-GMM$^{\mathrm{d}}$-GL and $\beta = 1/D$ for v-MFA-GL (denoted by dashed vertical lines) usually outperform the case of no hot posteriors ($\beta = 1$). Notably, $\beta = 0$ also achieves competitive results and even results in lower error rates than $\beta = 1$ for USPS, CIFAR-10 and SUSY. At $\beta = 0$, all non-zero entries of $\boldsymbol{Z}$ are assigned the same value. This implies that in the resulting graph ($\boldsymbol{W} = \boldsymbol{Z}^T\boldsymbol{Z}$), the information *which* components are connected is critical, whereas the connection strengths (graph weights) are not as important.

C.2.5 Ablation Study for Preprocessing

To evaluate the proposed v-GMM$^{\mathrm{d}}$-GL, v-MFA-GL and MFA-GL algorithms against established algorithms, we adopted the evaluation protocol of Zhang *et al.* (2023), which includes a standardized preprocessing pipeline for the USPS, MNIST, CIFAR-10 and EMNIST datasets (see Appendix C.1.2). For the USPS, MNIST and EMNIST datasets, dimensionality reduction was performed using PCA. In the case of CIFAR-10, feature extraction was first conducted using a pretrained ResNet-34 model; the resulting feature vectors were subsequently reduced to 30 dimensions via PCA.

In the following analysis, we examined the impact of this preprocessing (i.e., the application of PCA and the use of ResNet-34 features) on the classification error rates of the evaluated algorithms. Therefore, we repeated the experiments from Sec. 4.3.2 on the USPS, MNIST and EMNIST datasets without applying PCA as a preprocessing step. To avoid zero

variances at the image edges, we added standard Gaussian noise to the images. The results are depicted in Fig. C.3.

For the CIFAR-10 dataset, we conducted additional experiments to isolate the effects of each preprocessing step. Concretely, we evaluated three settings: no preprocessing, PCA applied directly to the image data, and the use of ResNet-34 feature extraction without subsequent PCA. The corresponding results are presented in Fig. C.4.

The results in Fig. C.3 indicate that preprocessing with PCA improves the classification error rates for algorithms based on GMMs with diagonal covariance matrices or MFAs. In contrast, the classification error rates for methods based on the k-means algorithm remain largely unchanged or exhibit slight improvements (compare, e.g., AGR, EAGR or f-FME on the USPS dataset in Fig. C.3).

The results in Fig. C.4 clearly show that all algorithms benefit substantially from ResNet-34 feature extraction. Compared to no preprocessing or only PCA, the classification error rate difference is approximately 30% to 50% across nearly all algorithms and experimental settings. Notably, for several algorithms including AGR, EAGR, f-FME, v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL, performance is even better when the methods are applied directly to ResNet-34 features, without additional PCA (compare also Fig. 4.5).

C.3 Additional Information on Results

In this section, we present supplementary details related to the results discussed in Sec. 4.3 in the main text. This includes a breakdown of the runtimes as well as a tabular summary of the observed error rates.

C.3.1 Additional Information on Runtimes

While Fig. 4.4 in the main text presents the overall runtimes of the algorithms, including both distribution learning (Dist. Learning) as well as graph learning and labeling (GLL), Tab. C.3 and Tab. C.4 provide a more detailed breakdown, detailing the runtimes for the two stages separately. These measurements were obtained by executing the algorithms in parallel (see Appendix C.1.6). Furthermore, we include the runtimes for GLL when the algorithms are executed on a single thread, as GLL^{1} in Tabs. C.3 and C.4.

DDGL as a Sparse Approach: Zhang *et al.* (2023) suggest that the transformation matrix $\boldsymbol{Z}$ can intuitively be considered sparse. To reduce the computational cost of the algorithms, very small values in the $\boldsymbol{Z}$ matrix have to be set to exactly zero. However, Zhang *et al.* (2023) did not specify the strategy they employed for this purpose. A commonly adopted heuristic involves introducing a threshold, whereby all values below this threshold are set to zero. Employing such a threshold can accelerate the graph learning and labeling stage of DDGL as well as MFA-GL. To show this effect, we executed the graph learning and labeling stage of DDGL and MFA-GL using a threshold of $1/C$. The distribution learning stage, naturally, is not affected by the threshold and remains unchanged. These variants are referred to as DDGL$^{(\mathrm{s})}$ or MFA-GL$^{(\mathrm{s})}$ in Tabs. C.3 and C.4. For the error rates, we observed no substantial changes to DDGL or MFA-GL with a dense transformation matrix $\boldsymbol{Z}$, unless the threshold caused the graph to become overly sparse. Concretely, setting the threshold too high led to an overly sparse transformation matrix $\boldsymbol{Z}$ and, consequently, an excessively sparse graph, which resulted in a decreased performance.

From the runtimes reported in Tabs. C.3 and C.4, it is evident that the distribution learning stage requires significantly longer runtimes than the graph learning and labeling stage across all methods and benchmark datasets (excluding GLL[1]). The v-GMM$^{\mathrm{d}}$-GL algorithm is consistently the fastest method (or tied with the fastest) in both distribution learning as well as graph learning and labeling throughout all datasets. When comparing DDGL$^{(\mathrm{s})}$ to DDGL and MFA-GL$^{(\mathrm{s})}$ to MFA-GL, we observe that sparsification can accelerate the graph learning and labeling stage. However, the impact on the total runtimes, including distribution learning, remains minor. Among all methods, MFA-GL (and MFA-GL$^{(\mathrm{s})}$), exhibits the slowest distribution learning stage, most notably on EMNIST, SUSY and MNIST8M (see Tab. C.4). These long runtimes can be attributed to the computationally expensive optimization of the MFA model using conventional EM. In contrast, the distribution learning time in v-MFA-GL, which utilizes the same distribution model, is substantially shorter, due to the variational optimization. Regarding graph learning and labeling, either f-FME or MFA-GL have the longest runtimes.

C.3.2 Additional Information on Error Rates

Table C.5 provides a summary of the error rates observed in the experiments discussed in Secs. 4.3.2 and 4.3.3 in the main text. These results are identical to those presented in Figs. 4.5 and 4.6 in the main text, but are here shown in tabular form.

Tab C.2: Comparison of error rates in % of distribution-based GSSL algorithms applied in this work, and of GSSL algorithms investigated in Li *et al.* (2020) on Letter, EMNIST and MNIST8M. Note that the error rates are here reported in terms of averages (and their standard deviations), while Tab. C.5 uses medians. No error margins were given in Li *et al.* (2020).

Dataset	N_l	AGR	EAGR	f-FME	r-FME	MiMoLaP$^{\mathrm{LG}}$	MiMoLaP$^{\mathrm{H}}$	DDGL	MFA-GL	v-GMM$^{\mathrm{d}}$-GL	v-MFA-GL	KMGS	GAG	MGS	HM	GCN_2	GCN_3	HAGR
Letter	26	68.58 ± 3.29	67.00 ± 3.50	68.14 ± 3.32	67.06 ± 3.58	69.16 ± 3.31	72.71 ± 3.39	71.68 ± 3.63	69.49 ± 3.87	65.46 ± 3.59	65.98 ± 3.51	–	–	–	–	–	–	–
	104	52.26 ± 1.84	47.82 ± 1.90	52.16 ± 1.94	48.22 ± 2.17	48.59 ± 2.09	50.11 ± 2.46	47.44 ± 2.43	45.66 ± 2.54	44.51 ± 2.20	41.41 ± 2.43	–	–	–	–	–	–	–
	130	–	–	–	–	–	–	–	–	–	–	59.89	54.43	43.96	49.93	54.79	64.84	45.67
	208	43.84 ± 1.63	37.59 ± 1.49	44.62 ± 1.71	39.33 ± 2.45	37.44 ± 1.67	38.71 ± 1.94	35.37 ± 1.99	33.96 ± 2.02	33.46 ± 1.71	29.35 ± 1.93	–	–	–	–	–	–	–
	260	–	–	–	–	–	–	–	–	–	–	55.41	44.55	33.87	40.46	46.03	58.12	37.26
	312	39.53 ± 1.39	32.28 ± 1.22	40.92 ± 1.50	37.51 ± 3.96	31.73 ± 1.38	33.10 ± 1.60	29.82 ± 1.54	28.49 ± 1.59	28.24 ± 1.39	23.82 ± 1.45	–	–	–	–	–	–	–
	390	–	–	–	–	–	–	–	–	–	–	50.23	38.70	30.91	35.52	42.13	56.04	32.35
	416	36.59 ± 1.22	28.71 ± 1.09	38.40 ± 1.32	40.44 ± 5.50	27.86 ± 1.22	29.40 ± 1.39	26.26 ± 1.30	25.30 ± 1.41	24.77 ± 1.18	20.70 ± 1.20	–	–	–	–	–	–	–
	520	34.36 ± 1.22	26.14 ± 0.98	36.48 ± 1.32	47.60 ± 6.62	25.29 ± 1.10	26.79 ± 1.23	23.97 ± 1.14	23.00 ± 1.23	22.45 ± 1.01	18.61 ± 0.98	–	–	–	–	–	–	–
EMNIST	10	18.92 ± 5.05	18.87 ± 5.17	19.16 ± 5.07	18.78 ± 5.20	38.80 ± 5.67	37.66 ± 8.90	31.83 ± 8.57	4.83 ± 4.56	17.69 ± 5.37	6.58 ± 3.96	–	–	–	–	–	–	–
	50	10.55 ± 1.74	9.24 ± 1.61	9.98 ± 1.72	9.23 ± 1.66	17.49 ± 2.59	14.93 ± 3.16	9.94 ± 2.26	2.48 ± 0.23	8.69 ± 1.60	2.94 ± 0.52	–	–	–	–	–	–	–
	100	8.84 ± 1.10	7.37 ± 0.88	8.27 ± 1.13	7.76 ± 1.01	12.43 ± 1.55	10.89 ± 1.99	7.37 ± 1.22	2.47 ± 0.16	6.87 ± 0.94	2.58 ± 0.22	50.77	21.94	4.96	5.20	15.94	26.18	8.14
	150	8.01 ± 0.82	6.58 ± 0.57	7.57 ± 0.85	7.41 ± 1.77	10.42 ± 1.18	9.22 ± 1.39	6.61 ± 0.79	2.49 ± 0.16	6.14 ± 0.59	2.46 ± 0.14	–	–	–	–	–	–	–
	200	7.54 ± 0.62	6.16 ± 0.42	7.22 ± 0.66	7.31 ± 1.49	9.42 ± 1.01	8.46 ± 1.17	6.36 ± 0.66	2.52 ± 0.16	5.75 ± 0.44	2.40 ± 0.10	48.55	15.66	4.92	5.05	13.20	15.79	6.83
	300	–	–	–	–	–	–	–	–	–	–	47.63	12.25	4.90	4.98	11.61	10.86	6.56
	400	–	–	–	–	–	–	–	–	–	–	47.78	10.93	4.88	4.95	10.33	9.68	6.08
	500	–	–	–	–	–	–	–	–	–	–	45.52	9.13	4.85	4.92	10.27	8.92	6.04
MNIST8M	10	27.25 ± 6.06	27.41 ± 6.30	27.55 ± 6.10	27.34 ± 6.38	49.95 ± 5.51	65.50 ± 9.32	52.27 ± 10.21	9.23 ± 5.41	27.32 ± 6.49	11.01 ± 4.99	–	–	–	–	–	–	–
	50	17.81 ± 1.73	16.10 ± 1.77	16.56 ± 1.70	16.33 ± 1.96	27.25 ± 2.83	25.49 ± 4.54	17.91 ± 2.49	4.45 ± 0.83	16.09 ± 1.87	5.31 ± 0.90	–	–	–	–	–	–	–
	100	15.78 ± 1.19	13.52 ± 1.20	14.21 ± 1.26	14.30 ± 1.67	20.78 ± 1.95	19.14 ± 2.28	14.11 ± 1.64	4.11 ± 0.29	13.35 ± 1.22	4.52 ± 0.37	–	–	–	–	–	–	–
	200	13.64 ± 0.77	11.37 ± 0.68	12.48 ± 0.91	13.24 ± 1.32	16.43 ± 1.20	15.61 ± 1.36	12.16 ± 0.98	4.08 ± 0.21	11.23 ± 0.72	4.11 ± 0.15	–	–	–	–	–	–	–
	300	12.55 ± 0.64	10.39 ± 0.52	11.78 ± 0.77	14.14 ± 3.06	14.72 ± 0.94	14.32 ± 1.04	11.62 ± 0.78	4.13 ± 0.20	10.28 ± 0.53	3.99 ± 0.11	–	–	–	–	–	–	–
	400	11.84 ± 0.53	9.77 ± 0.38	11.33 ± 0.68	16.91 ± 5.49	13.74 ± 0.74	13.46 ± 0.88	11.38 ± 0.64	4.18 ± 0.22	9.71 ± 0.41	3.91 ± 0.10	–	–	–	–	–	–	–
	500	11.44 ± 0.49	9.40 ± 0.33	11.09 ± 0.61	20.55 ± 6.99	13.13 ± 0.69	13.01 ± 0.76	11.32 ± 0.62	4.20 ± 0.20	9.37 ± 0.34	3.87 ± 0.08	–	–	–	–	–	–	–
	600	–	–	–	–	–	–	–	–	–	–	14.25	60.03	6.14	–	–	–	6.19
	700	–	–	–	–	–	–	–	–	–	–	13.30	59.88	5.52	–	–	–	5.82
	800	–	–	–	–	–	–	–	–	–	–	11.79	55.04	5.32	–	–	–	5.74
	900	–	–	–	–	–	–	–	–	–	–	11.55	54.01	5.27	–	–	–	5.71
	1000	–	–	–	–	–	–	–	–	–	–	11.23	50.68	4.89	–	–	–	5.64

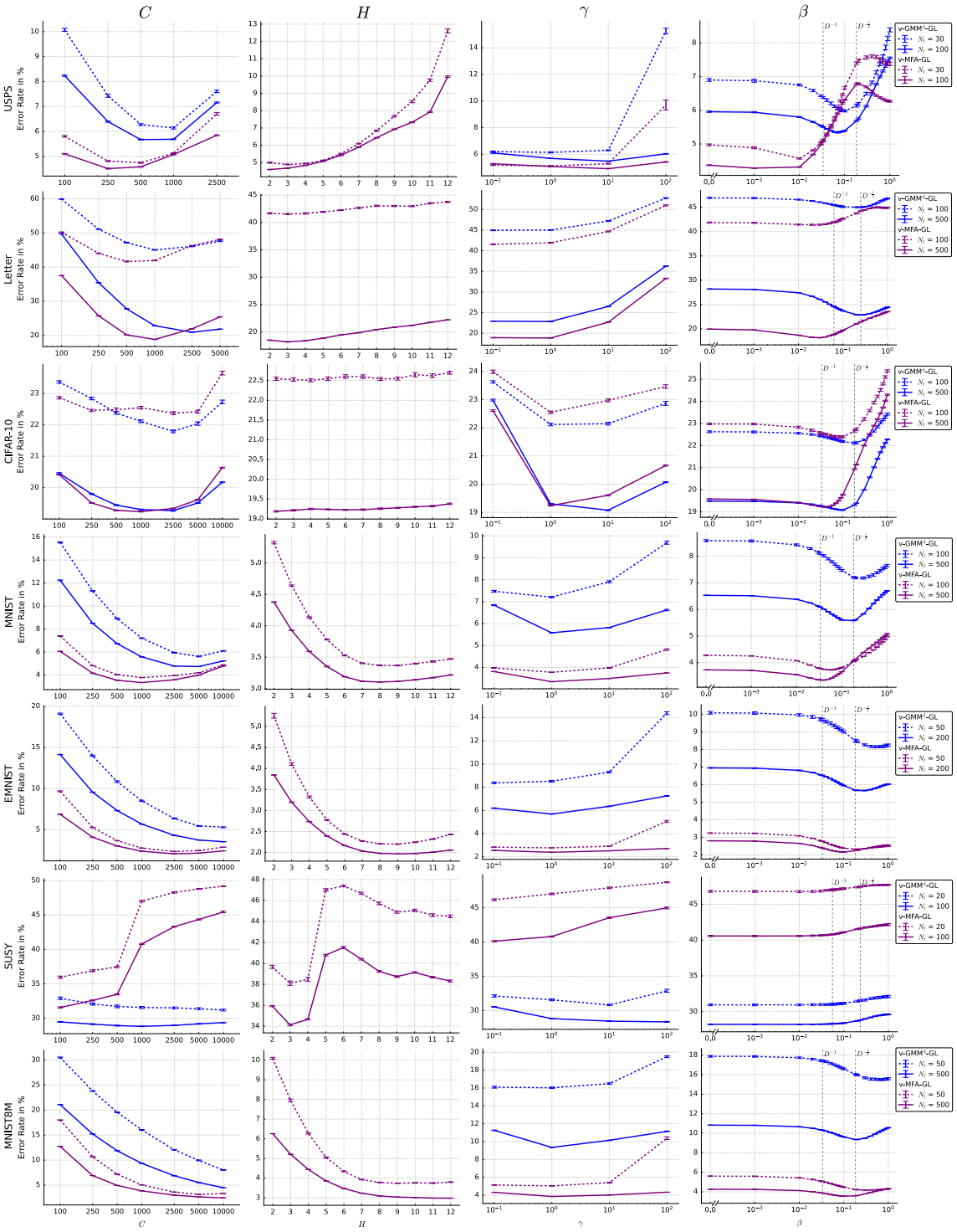

Fig C.2: Hyperparameter analysis for v-GMM$^{\mathrm{d}}$-GL and v-MFA-GL. Each row corresponds to one benchmark dataset. Each column illustrates the change of the error rate in % with one hyperparameter with all other hyperparameters held constant at their default values. The hyperplane dimensionality H is only relevant for v-MFA-GL. In the column for inverse temperature β, dashed vertical lines indicate the default values: $\beta = 1/\sqrt{D}$ for v-GMM$^{\mathrm{d}}$-GL and $\beta = 1/D$ for v-MFA-GL. Note that $\beta = 0$ is shifted to be visible on the logarithmic scale. Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling. Each legend corresponds to all subfigures in the respective row.

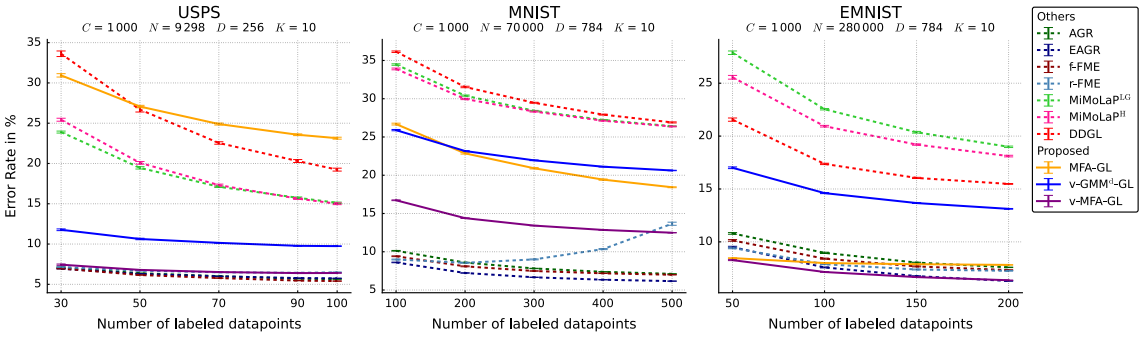

Fig C.3: Error rates in % for varying numbers of labeled data points of ten different GSSL methods on the USPS, MNIST and EMNIST datasets, without PCA preprocessing. The methods from the literature include AGR, EAGR, f-FME, r-FME, MiMoLaP$^{\mathrm{H}}$, MiMoLaP$^{\mathrm{LG}}$, and DDGL (dashed lines), while the newly proposed methods are MFA-GL, v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL (solid lines). Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling. The legend applies to all three subfigures.

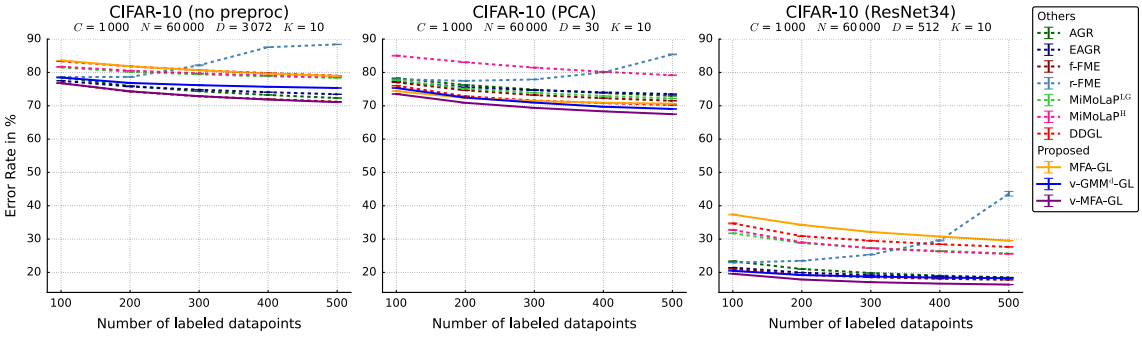

Fig C.4: Error rates in % for varying numbers of labeled data points of ten different GSSL methods on the CIFAR-10 dataset, comparing different preprocessing strategies: no preprocessing (left), PCA applied directly to the image data (mid), and the use of ResNet-34 feature extraction without subsequent PCA (right). The methods from the literature include AGR, EAGR, f-FME, r-FME, MiMoLaP$^{\mathrm{H}}$, MiMoLaP$^{\mathrm{LG}}$, and DDGL (dashed lines), while the newly proposed methods are MFA-GL, v-GMM$^{\mathrm{d}}$-GL, and v-MFA-GL (solid lines). Each error rate in the figure represents the median error rate calculated over 800 measurement points. The standard errors (error bars) in the median were calculated using bootstrap resampling. The legend applies to all three subfigures.

Tab C.3: Median runtime in seconds for the distribution learning (Dist. Learning) stage and graph learning and labeling (GLL) stage on USPS, Letter, CIFAR-10 and MNIST. GLL[1] denotes the time of the GLL stage when executed on a single thread. The lowest runtimes are highlighted in bold. The standard errors (error bars) in the median were calculated using bootstrap resampling.

	USPS			Letter			CIFAR-10			MNIST		
	Dist. Learning	GLL	GLL[1]	Dist. Learning	GLL	GLL[1]	Dist. Learning	GLL	GLL[1]	Dist. Learning	GLL	GLL[1]
AGR	0.63 ± 0.01	0.03 ± 0.01	0.09 ± 0.01	0.47 ± 0.01	0.05 ± 0.01	0.16 ± 0.01	1.05 ± 0.01	0.10 ± 0.01	0.48 ± 0.01	1.26 ± 0.01	0.10 ± 0.01	0.54 ± 0.01
EAGR	0.63 ± 0.01	0.03 ± 0.01	0.07 ± 0.01	0.47 ± 0.01	0.05 ± 0.01	0.13 ± 0.01	1.05 ± 0.01	0.07 ± 0.01	0.32 ± 0.01	1.26 ± 0.01	0.08 ± 0.01	0.37 ± 0.01
f-FME	0.63 ± 0.01	0.06 ± 0.01	0.11 ± 0.01	0.47 ± 0.01	0.09 ± 0.01	0.19 ± 0.01	1.05 ± 0.01	0.30 ± 0.01	0.67 ± 0.01	1.26 ± 0.01	0.34 ± 0.01	0.78 ± 0.01
r-FME	0.63 ± 0.01	0.03 ± 0.01	0.07 ± 0.01	0.47 ± 0.01	0.04 ± 0.01	0.12 ± 0.01	1.05 ± 0.01	0.07 ± 0.01	0.32 ± 0.01	1.26 ± 0.01	0.07 ± 0.01	0.37 ± 0.01
MiMoLaP$^{\text{LG}}$	0.35 ± 0.01	0.02 ± 0.01	0.06 ± 0.01	0.64 ± 0.01	0.02 ± 0.01	0.11 ± 0.01	1.35 ± 0.01	0.02 ± 0.01	0.14 ± 0.01	1.02 ± 0.01	0.02 ± 0.01	0.15 ± 0.01
MiMoLaP$^{\text{H}}$	0.35 ± 0.01	0.02 ± 0.01	0.06 ± 0.01	0.64 ± 0.01	0.02 ± 0.01	0.10 ± 0.01	1.35 ± 0.01	0.02 ± 0.01	0.13 ± 0.01	1.02 ± 0.01	0.02 ± 0.01	0.15 ± 0.01
DDGL	0.35 ± 0.01	0.05 ± 0.01	0.40 ± 0.01	0.64 ± 0.01	0.09 ± 0.01	0.89 ± 0.01	1.35 ± 0.01	0.22 ± 0.01	2.37 ± 0.01	1.02 ± 0.01	0.26 ± 0.01	2.75 ± 0.01
DDGL$^{(\text{s})}$	0.35 ± 0.01	0.02 ± 0.01	0.04 ± 0.01	0.64 ± 0.01	0.03 ± 0.01	0.06 ± 0.01	1.35 ± 0.01	0.06 ± 0.01	0.16 ± 0.01	1.02 ± 0.01	0.05 ± 0.01	0.16 ± 0.01
MFA-GL	4.64 ± 0.01	0.06 ± 0.01	0.44 ± 0.01	20.90 ± 0.08	0.12 ± 0.01	0.90 ± 0.01	39.59 ± 0.05	0.28 ± 0.01	2.37 ± 0.01	33.17 ± 0.01	0.32 ± 0.01	2.79 ± 0.01
MFA-GL$^{(\text{s})}$	4.64 ± 0.01	0.02 ± 0.01	0.04 ± 0.01	20.90 ± 0.08	0.03 ± 0.01	0.06 ± 0.01	39.59 ± 0.05	0.04 ± 0.01	0.14 ± 0.01	33.17 ± 0.01	0.04 ± 0.01	0.15 ± 0.01
v-GMM$^{\text{d}}$-GL	$\mathbf{0.29 \pm 0.01}$	$\mathbf{< 0.01}$	$\mathbf{< 0.01}$	$\mathbf{0.23 \pm 0.01}$	$\mathbf{< 0.01}$	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.84 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.04 \pm 0.01}$	$\mathbf{0.60 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.04 \pm 0.01}$
v-MFA-GL	0.45 ± 0.01	$\mathbf{< 0.01}$	0.02 ± 0.01	1.02 ± 0.01	$\mathbf{< 0.01}$	0.03 ± 0.01	2.28 ± 0.01	$\mathbf{0.01 \pm 0.01}$	0.08 ± 0.01	1.79 ± 0.01	$\mathbf{0.01 \pm 0.01}$	0.09 ± 0.01

Tab C.4: Median runtime in seconds for the distribution learning (Dist. Learning) stage and graph learning and labeling (GLL) stage on EMNIST, SUSY, and MNIST8M. GLL[1] denotes the time of the GLL stage when executed on a single thread. The lowest runtimes are highlighted in bold. The standard errors (error bars) in the median were calculated using bootstrap resampling.

	EMNIST			SUSY			MNIST8M		
	Dist. Learning	GLL	GLL[1]	Dist. Learning	GLL	GLL[1]	Dist. Learning	GLL	GLL[1]
AGR	8.25 ± 0.02	0.29 ± 0.01	2.05 ± 0.01	162.20 ± 0.04	6.18 ± 0.04	30.97 ± 0.01	325.31 ± 0.01	8.66 ± 0.01	59.35 ± 0.01
EAGR	8.25 ± 0.02	0.22 ± 0.01	1.41 ± 0.01	162.20 ± 0.04	5.57 ± 0.04	22.71 ± 0.01	325.31 ± 0.01	7.21 ± 0.01	41.50 ± 0.01
f-FME	8.25 ± 0.02	1.48 ± 0.01	3.20 ± 0.01	162.20 ± 0.04	18.74 ± 0.02	44.52 ± 0.01	325.31 ± 0.01	49.69 ± 0.01	99.76 ± 0.01
r-FME	8.25 ± 0.02	0.23 ± 0.01	1.42 ± 0.01	162.20 ± 0.04	4.26 ± 0.01	23.35 ± 0.01	325.31 ± 0.01	7.69 ± 0.01	42.17 ± 0.01
MiMoLaP$^{\text{LG}}$	3.11 ± 0.01	0.05 ± 0.01	0.48 ± 0.01	106.48 ± 0.05	0.42 ± 0.01	6.31 ± 0.01	92.60 ± 0.14	0.81 ± 0.01	12.56 ± 0.01
MiMoLaP$^{\text{H}}$	3.11 ± 0.01	0.04 ± 0.01	0.48 ± 0.01	106.48 ± 0.05	0.40 ± 0.01	6.14 ± 0.01	92.60 ± 0.14	0.80 ± 0.01	12.53 ± 0.01
DDGL	3.11 ± 0.01	1.01 ± 0.01	10.81 ± 0.01	106.48 ± 0.05	17.79 ± 0.03	196.37 ± 0.12	92.60 ± 0.14	30.38 ± 0.01	309.68 ± 0.07
DDGL$^{(\text{s})}$	3.11 ± 0.01	0.12 ± 0.01	0.57 ± 0.01	106.48 ± 0.05	8.38 ± 0.02	17.94 ± 0.03	92.60 ± 0.14	7.58 ± 0.03	20.02 ± 0.01
MFA-GL	130.71 ± 0.01	1.12 ± 0.01	10.85 ± 0.01	18544.58 ± 6.53	22.26 ± 0.03	202.25 ± 0.24	3806.99 ± 3.35	32.15 ± 0.03	311.45 ± 0.01
MFA-GL$^{(\text{s})}$	130.71 ± 0.01	0.12 ± 0.01	0.56 ± 0.01	18544.58 ± 6.53	5.77 ± 0.04	14.39 ± 0.03	3806.99 ± 3.35	5.69 ± 0.04	18.03 ± 0.01
v-GMM$^{\text{d}}$-GL	$\mathbf{1.56 \pm 0.01}$	$\mathbf{0.03 \pm 0.01}$	$\mathbf{0.22 \pm 0.01}$	$\mathbf{27.33 \pm 0.01}$	$\mathbf{0.37 \pm 0.01}$	$\mathbf{3.96 \pm 0.01}$	$\mathbf{31.96 \pm 0.01}$	$\mathbf{0.75 \pm 0.01}$	$\mathbf{8.21 \pm 0.01}$
v-MFA-GL	6.07 ± 0.01	$\mathbf{0.03 \pm 0.01}$	0.42 ± 0.01	617.24 ± 0.68	0.54 ± 0.01	8.71 ± 0.03	179.79 ± 0.03	1.07 ± 0.01	16.64 ± 0.02

Tab C.5: Median error rates in % of various GSSL methods for different numbers of labeled data points N_l on seven datasets. The lowest error rates are highlighted in bold. The standard errors (error bars) in the median were calculated using bootstrap resampling.

Dataset	N_l	AGR	EAGR	f-FME	r-FME	MiMoLaP$^{\text{LG}}$	MiMoLaP$^{\text{H}}$	DDGL	MFA-GL	v-GMM$^{\text{d}}$-GL	v-MFA-GL
USPS	10	10.82 ± 0.15	9.86 ± 0.10	10.87 ± 0.15	9.80 ± 0.11	33.50 ± 0.20	40.45 ± 0.27	41.50 ± 0.32	29.08 ± 0.31	8.06 ± 0.18	$\mathbf{5.82 \pm 0.07}$
	30	7.50 ± 0.04	6.92 ± 0.04	7.37 ± 0.03	7.10 ± 0.04	18.67 ± 0.13	19.08 ± 0.19	20.20 ± 0.25	12.82 ± 0.16	6.14 ± 0.03	$\mathbf{5.13 \pm 0.01}$
	50	6.80 ± 0.02	6.28 ± 0.02	6.57 ± 0.03	6.61 ± 0.03	14.32 ± 0.09	13.12 ± 0.16	14.22 ± 0.11	10.09 ± 0.08	5.80 ± 0.02	$\mathbf{5.09 \pm 0.01}$
	70	6.44 ± 0.02	6.07 ± 0.02	6.18 ± 0.02	6.59 ± 0.02	12.33 ± 0.05	10.54 ± 0.12	11.55 ± 0.10	8.54 ± 0.06	5.74 ± 0.02	$\mathbf{5.09 \pm 0.01}$
	90	6.19 ± 0.02	5.89 ± 0.02	5.90 ± 0.01	6.56 ± 0.03	10.96 ± 0.05	9.25 ± 0.09	10.35 ± 0.06	8.20 ± 0.08	5.67 ± 0.02	$\mathbf{5.10 \pm 0.01}$
	100	6.09 ± 0.01	5.84 ± 0.01	5.83 ± 0.01	6.51 ± 0.02	10.50 ± 0.05	8.81 ± 0.08	10.00 ± 0.05	7.81 ± 0.06	5.69 ± 0.01	$\mathbf{5.08 \pm 0.01}$
Letter	26	68.44 ± 0.09	67.11 ± 0.10	68.07 ± 0.08	67.14 ± 0.08	69.28 ± 0.11	72.58 ± 0.11	71.53 ± 0.13	69.50 ± 0.12	$\mathbf{65.16 \pm 0.11}$	66.04 ± 0.14
	104	52.18 ± 0.07	47.71 ± 0.07	52.08 ± 0.05	48.00 ± 0.05	48.51 ± 0.06	49.96 ± 0.06	47.26 ± 0.07	45.60 ± 0.09	44.41 ± 0.07	$\mathbf{41.33 \pm 0.06}$
	208	43.77 ± 0.06	37.58 ± 0.04	44.55 ± 0.07	38.96 ± 0.07	37.40 ± 0.04	38.76 ± 0.07	35.29 ± 0.05	33.84 ± 0.06	33.47 ± 0.05	$\mathbf{29.21 \pm 0.05}$
	312	39.50 ± 0.05	32.29 ± 0.03	40.89 ± 0.04	36.55 ± 0.12	31.70 ± 0.04	33.09 ± 0.03	29.76 ± 0.05	28.45 ± 0.05	28.19 ± 0.04	$\mathbf{23.85 \pm 0.04}$
	416	36.60 ± 0.03	28.68 ± 0.03	38.39 ± 0.04	39.76 ± 0.16	27.82 ± 0.03	29.38 ± 0.06	26.24 ± 0.04	25.30 ± 0.05	24.71 ± 0.03	$\mathbf{20.72 \pm 0.03}$
	520	34.36 ± 0.04	26.10 ± 0.03	36.52 ± 0.04	47.59 ± 0.21	25.27 ± 0.03	26.79 ± 0.04	23.93 ± 0.04	22.98 ± 0.04	22.39 ± 0.04	$\mathbf{18.55 \pm 0.03}$
CIFAR-10	10	37.87 ± 0.25	36.94 ± 0.21	37.90 ± 0.23	36.62 ± 0.22	42.49 ± 0.29	63.67 ± 0.29	62.44 ± 0.23	40.43 ± 0.25	$\mathbf{35.81 \pm 0.25}$	36.46 ± 0.21
	100	25.66 ± 0.05	23.26 ± 0.04	23.55 ± 0.05	24.54 ± 0.05	26.59 ± 0.04	25.78 ± 0.07	24.75 ± 0.04	25.45 ± 0.04	$\mathbf{22.12 \pm 0.03}$	22.55 ± 0.03
	200	23.64 ± 0.02	21.98 ± 0.02	21.75 ± 0.02	25.01 ± 0.03	24.71 ± 0.02	23.87 ± 0.02	23.34 ± 0.03	24.60 ± 0.02	$\mathbf{20.69 \pm 0.02}$	20.88 ± 0.02
	300	22.41 ± 0.02	21.32 ± 0.02	21.01 ± 0.02	26.07 ± 0.03	23.92 ± 0.03	23.30 ± 0.03	23.15 ± 0.02	24.41 ± 0.02	$\mathbf{20.02 \pm 0.01}$	$\mathbf{20.02 \pm 0.02}$
	400	21.61 ± 0.01	20.91 ± 0.02	20.56 ± 0.02	28.19 ± 0.05	23.44 ± 0.02	23.17 ± 0.02	23.24 ± 0.02	24.37 ± 0.02	19.62 ± 0.02	$\mathbf{19.57 \pm 0.01}$
	500	21.06 ± 0.01	20.58 ± 0.01	20.23 ± 0.02	31.99 ± 0.09	23.07 ± 0.02	23.05 ± 0.02	23.19 ± 0.02	24.39 ± 0.02	19.31 ± 0.01	$\mathbf{19.24 \pm 0.01}$
MNIST	10	18.80 ± 0.15	18.68 ± 0.13	19.08 ± 0.12	18.62 ± 0.13	40.54 ± 0.22	42.14 ± 0.25	36.56 ± 0.25	$\mathbf{6.45 \pm 0.09}$	17.18 ± 0.13	7.89 ± 0.11
	100	9.70 ± 0.04	8.15 ± 0.03	8.95 ± 0.04	8.61 ± 0.03	13.67 ± 0.04	11.19 ± 0.07	8.11 ± 0.06	4.95 ± 0.01	7.21 ± 0.02	$\mathbf{3.78 \pm 0.01}$
	200	8.37 ± 0.02	7.02 ± 0.01	7.81 ± 0.02	8.39 ± 0.02	10.44 ± 0.04	8.86 ± 0.04	7.24 ± 0.02	5.00 ± 0.01	6.21 ± 0.02	$\mathbf{3.54 \pm 0.01}$
	300	7.68 ± 0.01	6.55 ± 0.01	7.31 ± 0.02	8.78 ± 0.03	9.18 ± 0.02	8.14 ± 0.03	7.06 ± 0.01	5.01 ± 0.01	5.88 ± 0.01	$\mathbf{3.44 \pm 0.01}$
	400	7.32 ± 0.01	6.28 ± 0.01	7.07 ± 0.01	9.80 ± 0.04	8.59 ± 0.03	7.88 ± 0.02	7.07 ± 0.01	5.03 ± 0.01	5.69 ± 0.01	$\mathbf{3.38 \pm 0.01}$
	500	7.08 ± 0.01	6.11 ± 0.01	6.90 ± 0.01	12.32 ± 0.08	8.23 ± 0.02	7.73 ± 0.01	7.14 ± 0.02	5.06 ± 0.01	5.58 ± 0.01	$\mathbf{3.35 \pm 0.01}$
EMNIST	10	18.23 ± 0.08	18.09 ± 0.14	18.45 ± 0.10	18.02 ± 0.13	38.37 ± 0.13	37.06 ± 0.32	30.86 ± 0.25	$\mathbf{2.76 \pm 0.03}$	16.65 ± 0.12	5.35 ± 0.10
	50	10.33 ± 0.08	9.01 ± 0.06	9.81 ± 0.06	9.04 ± 0.05	17.23 ± 0.08	14.66 ± 0.10	9.73 ± 0.06	$\mathbf{2.44 \pm 0.01}$	8.51 ± 0.05	2.77 ± 0.01
	100	8.72 ± 0.03	7.24 ± 0.02	8.12 ± 0.03	7.64 ± 0.04	12.34 ± 0.04	10.67 ± 0.06	7.18 ± 0.03	$\mathbf{2.45 \pm 0.01}$	6.73 ± 0.03	2.53 ± 0.01
	150	7.87 ± 0.02	6.53 ± 0.01	7.41 ± 0.02	7.21 ± 0.03	10.36 ± 0.03	8.99 ± 0.04	6.50 ± 0.02	2.48 ± 0.01	6.05 ± 0.02	$\mathbf{2.44 \pm 0.01}$
	200	7.45 ± 0.02	6.11 ± 0.01	7.12 ± 0.02	7.09 ± 0.02	9.35 ± 0.03	8.33 ± 0.03	6.30 ± 0.03	2.50 ± 0.01	5.69 ± 0.01	$\mathbf{2.39 \pm 0.01}$
SUSY	2	48.33 ± 0.13	48.34 ± 0.11	48.27 ± 0.13	48.33 ± 0.12	40.58 ± 0.19	45.70 ± 0.01	45.71 ± 0.01	49.82 ± 0.02	$\mathbf{34.87 \pm 0.36}$	49.86 ± 0.03
	20	41.30 ± 0.17	41.33 ± 0.15	40.83 ± 0.15	40.72 ± 0.11	34.94 ± 0.16	39.23 ± 0.10	36.39 ± 0.13	47.89 ± 0.08	$\mathbf{31.56 \pm 0.10}$	47.02 ± 0.09
	40	38.35 ± 0.15	38.14 ± 0.11	37.39 ± 0.17	37.74 ± 0.10	33.04 ± 0.12	34.40 ± 0.11	31.11 ± 0.08	45.85 ± 0.08	$\mathbf{30.34 \pm 0.06}$	44.60 ± 0.08
	60	36.37 ± 0.10	36.22 ± 0.10	35.56 ± 0.11	36.10 ± 0.08	31.66 ± 0.08	31.73 ± 0.09	$\mathbf{29.41 \pm 0.04}$	44.25 ± 0.08	29.53 ± 0.05	42.83 ± 0.08
	80	35.40 ± 0.11	35.09 ± 0.07	34.34 ± 0.13	35.26 ± 0.06	31.21 ± 0.06	30.37 ± 0.06	$\mathbf{28.89 \pm 0.04}$	43.12 ± 0.06	29.01 ± 0.04	41.87 ± 0.08
	100	34.32 ± 0.11	34.23 ± 0.06	33.16 ± 0.12	34.83 ± 0.06	30.89 ± 0.04	29.76 ± 0.05	$\mathbf{28.63 \pm 0.05}$	42.26 ± 0.07	28.82 ± 0.03	40.79 ± 0.06
MNIST8M	10	25.90 ± 0.17	26.12 ± 0.18	26.27 ± 0.19	26.06 ± 0.16	49.84 ± 0.15	65.86 ± 0.34	51.67 ± 0.26	$\mathbf{7.40 \pm 0.09}$	26.36 ± 0.20	9.65 ± 0.14
	50	17.80 ± 0.05	15.92 ± 0.05	16.50 ± 0.04	16.18 ± 0.06	27.12 ± 0.09	24.63 ± 0.11	17.67 ± 0.08	$\mathbf{4.17 \pm 0.01}$	16.03 ± 0.06	5.05 ± 0.02
	100	15.72 ± 0.03	13.43 ± 0.03	14.11 ± 0.03	14.17 ± 0.04	20.75 ± 0.06	18.88 ± 0.05	13.99 ± 0.05	$\mathbf{4.05 \pm 0.01}$	13.24 ± 0.04	4.43 ± 0.01
	200	13.58 ± 0.03	11.33 ± 0.02	12.37 ± 0.03	13.12 ± 0.04	16.37 ± 0.03	15.54 ± 0.04	12.08 ± 0.03	$\mathbf{4.06 \pm 0.01}$	11.16 ± 0.02	4.08 ± 0.01
	300	12.50 ± 0.02	10.33 ± 0.01	11.69 ± 0.02	13.50 ± 0.04	14.68 ± 0.03	14.29 ± 0.04	11.55 ± 0.03	4.11 ± 0.01	10.20 ± 0.01	$\mathbf{3.98 \pm 0.01}$
	400	11.78 ± 0.02	9.74 ± 0.01	11.27 ± 0.02	14.99 ± 0.08	13.68 ± 0.03	13.43 ± 0.04	11.33 ± 0.02	4.16 ± 0.01	9.67 ± 0.01	$\mathbf{3.91 \pm 0.01}$
	500	11.38 ± 0.01	9.37 ± 0.01	11.01 ± 0.02	18.13 ± 0.10	13.11 ± 0.02	13.00 ± 0.02	11.34 ± 0.02	4.19 ± 0.01	9.35 ± 0.01	$\mathbf{3.86 \pm 0.01}$

Acknowledgments. During the preparation of this work the authors partly used ChatGPT[21] and DeepL[22] 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.

[21] `https://chatgpt.com/`
[22] `https://www.deepl.com/`

Appendix D

Supplementary Information on Variational Zero-Shot Denoising using Mixtures of Factor Analyzers

D.1 Details on the Data Estimator for MFAs

In this section, we derive the probabilistic data estimator for the mixture of factor analyzers (MFA) model, which is used in the denoising experiments presented in Ch. 5. Recall that the generative model of mixtures of factor analyzers (MFAs; Ghahramani and Hinton, 1996; Richardson and Weiss, 2018) is given by:

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

The probabilistic data estimator is based on the posterior predictive distribution $p(\boldsymbol{x}^{\text{est}}|\boldsymbol{x})$ (Drefs *et al.*, 2022), which, for the MFA model, is given by

$$p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) = \sum_c \int p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{z}, \boldsymbol{\Theta}) p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) d\boldsymbol{z}. \quad \text{(D.2)}$$

Denoising is then performed by replacing the observed noisy pixel values with the expectation values of this posterior predictive distribution:

$$\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})} \left[\boldsymbol{x}^{\text{est}} \right] = \mathbb{E}_{p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})} \left[\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{z}, \boldsymbol{\Theta})} \left[\boldsymbol{x}^{\text{est}} \right] \right]. \quad \text{(D.3)}$$

The inner expectation on the right-hand side of Eq. (D.3) corresponds to the first moment of the noise model (see Eq. (D.1), right), given by $\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{z}, \boldsymbol{\Theta})} \left[\boldsymbol{x}^{\text{est}} \right] = \boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c$. Substituting this into the equation yields:

$$\begin{aligned} \mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})} \left[\boldsymbol{x}^{\text{est}} \right] &= \mathbb{E}_{p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})} \left[\boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c \right] \\ &= \sum_c \int p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) (\boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c) d\boldsymbol{z} \end{aligned} \quad \text{(D.4)}$$

The joint posterior distribution $p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})$ can be factorized as $p(c, \boldsymbol{z} \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) = p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) p(\boldsymbol{z} \,|\, c, \boldsymbol{x}, \boldsymbol{\Theta})$. Inserting this in Eq. (D.4) yields:

$$\begin{aligned}
\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{x}^{\text{est}}\big] &= \sum_c \int p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) p(\boldsymbol{z} \,|\, c, \boldsymbol{x}, \boldsymbol{\Theta}) (\boldsymbol{\Lambda}_c \boldsymbol{z} + \boldsymbol{\mu}_c) d\boldsymbol{z} \\
&= \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\Lambda}_c \int p(\boldsymbol{z} \,|\, c, \boldsymbol{x}, \boldsymbol{\Theta})\, \boldsymbol{z}\, d\boldsymbol{z} + \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\mu}_c \\
&= \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\Lambda}_c \mathbb{E}_{p(\boldsymbol{z} \,|\, c, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{z}\big] + \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\mu}_c
\end{aligned} \tag{D.5}$$

The posterior $p(\boldsymbol{z} \,|\, c, \boldsymbol{x}, \boldsymbol{\Theta})$ conforms to a Gaussian distribution $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ with $\boldsymbol{\Sigma}_z = (\boldsymbol{I} + \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} \boldsymbol{\Lambda}_c)^{-1} = \boldsymbol{L}_c^{-1}$ and $\boldsymbol{\mu}_z = \boldsymbol{\Sigma}_z \boldsymbol{\Lambda}_c^\top \boldsymbol{D}_c^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_c) = \boldsymbol{V}_c (\boldsymbol{x} - \boldsymbol{\mu}_c)$. Consequently, the expectation of $\boldsymbol{z}$ w.r.t. $p(\boldsymbol{z} \,|\, c, \boldsymbol{x}_n, \tilde{\boldsymbol{\Theta}})$ is $\boldsymbol{\mu}_z$. Inserting this into Eq. (D.5) yields:

$$\begin{aligned}
\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{x}^{\text{est}}\big] &= \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\Lambda}_c \, \boldsymbol{\mu}_z + \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\mu}_c \\
&= \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\Lambda}_c \boldsymbol{V}_c (\boldsymbol{x} - \boldsymbol{\mu}_c) + \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) \boldsymbol{\mu}_c \\
&= \sum_c p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}) (\boldsymbol{\Lambda}_c \boldsymbol{V}_c (\boldsymbol{x} - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c) \\
&= \mathbb{E}_{p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{\Lambda}_c \boldsymbol{V}_c (\boldsymbol{x} - \boldsymbol{\mu}_c) + \boldsymbol{\mu}_c\big].
\end{aligned} \tag{D.6}$$

D.2 Details on Denoising with diagonal GMMs

In this section, we describe how a single noisy image can be denoised using Gaussian mixture models (GMMs) with diagonal covariance matrices. The procedure is analogous to the denoising approach based on MFAs described in Sec. 5.2.1. For completeness, we derive the corresponding data estimator for this model below. Recall that the generative model of GMMs with diagonal covariances is given by:

$$p(c \,|\, \boldsymbol{\Theta}) = \pi_c, \qquad p(\boldsymbol{x} \,|\, c, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{D}_c), \tag{D.7}$$

where $\pi_c \in [0, 1]$ are the mixing proportions, $\boldsymbol{\mu}_c \in \mathbb{R}^D$ are the means and $\boldsymbol{D}_c = \text{diag}(\sigma_{c1}^2, ..., \sigma_{cD}^2)$ is the diagonal covariance matrix for component c. The GMM is then optimized using the variational EM algorithm, analogously to MFA optimization, as detailed in Ch. 3 and Alg. 2.

The probabilistic data estimator is based on the posterior predictive distribution $p(\boldsymbol{x}^{\text{est}} | \boldsymbol{x})$ (Drefs *et al.*, 2022), which, for this GMM model, is given by

$$p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}) = \sum_c p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{\Theta}) p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta}). \tag{D.8}$$

The non-noisy pixel values are then obtained as the expectation of the posterior predictive distribution, as follows:

$$\begin{aligned}
\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{x}^{\text{est}}\big] &= \mathbb{E}_{p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\mathbb{E}_{p(\boldsymbol{x}^{\text{est}} \,|\, c, \boldsymbol{\Theta})}\big[\boldsymbol{x}^{\text{est}}\big]\big] \\
&= \mathbb{E}_{p(c \,|\, \boldsymbol{x}, \boldsymbol{\Theta})}\big[\boldsymbol{\mu}_c\big]
\end{aligned} \tag{D.9}$$

D.3 Additional Results

Additional results related to the experiments in Sec. 5.3 of the main text are provided in Tab. D.1. This table reports the accuracy of each DNN-based method when the maximum number of iterations is limited so that each algorithm runs only as long as Noise2Fast (N2F) requires to fully denoise the images. The accuracies of v-GMM$^{\mathrm{d}}$ and v-MFA are unaffected by this limitation, as both methods are faster than N2F.

Tab D.1: PSNR, SSIM and average runtime per image (in seconds) for eight blind zero-shot denoising algorithms evaluated on the Set12 and BSD68 datasets for different noise levels σ. Accuracy is reported at the time point when Noise2Fast completes its denoising (except for v-GMM$^{\mathrm{d}}$ and v-MFA which are faster). Results for N2S, N2V, DIP3000, Ne2Ne, S2S and N2F are taken from Lequyer *et al.* (2022b). Each value (PSNR, SSIM and runtime) represents the mean over all images in the corresponding dataset. For v-GMM$^{\mathrm{d}}$ and v-MFA, the standard error of the mean (SEM) is also reported. Error metrics for the remaining methods were not provided in Lequyer *et al.* (2022b). Note that the SSIM values are scaled by a factor of 10.

Dataset	σ		N2S	N2V	DIP3000	Ne2Ne	S2S	N2F	v-GMM$^{\mathrm{d}}$	v-MFA
Set12	15	PSNR	26.68	26.09	28.20	25.01	27.03	**31.10**	26.91 ± 0.60	30.58 ± 0.50
		SSIM	7.91	7.30	8.01	6.96	7.99	**8.71**	7.90 ± 0.15	8.67 ± 0.10
		Time per Image in s	22.00	22.00	22.00	22.00	22.00	22.00	**0.88 ± 0.15**	3.71 ± 0.76
	25	PSNR	25.72	24.48	26.53	24.07	25.83	29.05	26.62 ± 0.55	**29.16 ± 0.41**
		SSIM	7.20	6.47	7.35	5.83	7.23	8.22	7.78 ± 0.14	**8.26 ± 0.10**
		Time per Image in s	18.00	18.00	18.00	18.00	18.00	18.00	**0.83 ± 0.14**	3.39 ± 0.67
	35	PSNR	24.10	22.01	24.95	21.88	25.10	27.57	26.16 ± 0.50	**27.82 ± 0.36**
		SSIM	6.04	5.00	6.50	4.78	6.52	**7.81**	7.59 ± 0.14	7.76 ± 0.10
		Time per Image in s	19.00	19.00	19.00	19.00	19.00	19.00	**0.81 ± 0.13**	3.28 ± 0.61
	50	PSNR	23.32	21.98	22.28	21.23	24.21	25.82	25.32 ± 0.46	**26.13 ± 0.34**
		SSIM	5.71	4.99	5.04	3.94	6.10	7.23	**7.25 ± 0.15**	7.04 ± 0.11
		Time per Image in s	21.00	21.00	21.00	21.00	21.00	21.00	**0.79 ± 0.12**	3.42 ± 0.63
BSD68	25	PSNR	25.48	22.91	25.62	22.20	25.32	**28.12**	25.38 ± 0.42	27.25 ± 0.40
		SSIM	7.03	6.53	6.91	5.47	6.91	**7.89**	6.71 ± 0.16	7.47 ± 0.11
		Time per Image in s	29.00	29.00	29.00	29.00	29.00	29.00	**0.80 ± 0.01**	3.58 ± 0.05
	50	PSNR	23.56	22.42	22.08	20.62	23.83	25.23	24.50 ± 0.37	**25.30 ± 0.32**
		SSIM	6.01	4.91	4.83	3.65	6.33	**6.70**	6.31 ± 0.15	6.52 ± 0.11
		Time per Image in s	26.00	26.00	26.00	26.00	26.00	26.00	**0.81 ± 0.01**	3.80 ± 0.06

Publication List

Peer-Reviewed Journal Articles and Journal Articles under Review

Sebastian Salwig*, Jakob Drefs* and Jörg Lücke. Zero-shot denoising of microscopy images recorded at high-resolution limits. *PLOS Computational Biology*, 20(6):e1012192, 2024. *Joint first authorship.

Sebastian Salwig*, Till Kahlke*, Florian Hirschberger, Dennis Forster and Jörg Lücke. Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters. 2025. Under Review. *Joint first authorship.

Sebastian Salwig*, Till Kahlke* and Jörg Lücke. Variational Graph-Based Semi-Supervised Learning. 2025. Under Review. *Joint first authorship.

Malte Voskamp, Florian Büther, Sebastian Salwig, Martin Mamach, Florian Wilke, Johannes Stein, Andreas Büchner, Tobias Ross, Lilli Geworski, Frank M. Bengel, Thomas Lenarz, Jörg Lücke and Georg Berding. [^{15}O]H_2O PET auditory activation studies – improved results and possible dose reduction with Gaussian mixture model denoising. 2025. Under Review.

Preprints

Jakob Drefs, Sebastian Salwig and Jörg Lücke. Visualization of SARS-CoV-2 Infection Scenes by 'Zero-Shot' Enhancements of Electron Microscopy Images. *bioRxiv* 2021.02.25.432265, 2021.

Sebastian Salwig*, Till Kahlke*, Florian Hirschberger, Dennis Forster and Jörg Lücke. Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters. arXiv preprint arXiv:2501.12299, 2025. *Joint first authorship.

Declaration of AI-assisted Technologies in the Writing Process

During the preparation of this thesis the authors partly used ChatGPT[23] and DeepL[24] 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 this thesis.

[23] https://chatgpt.com/
[24] https://www.deepl.com/

Bibliography

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya *et al.* GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* 2023.

M. M. Ahsan, S. A. Luna and Z. Siddique. Machine-Learning-Based Disease Diagnosis: A Comprehensive Review. Healthcare, volume 10, 541. 2022.

R. Alrawili, A. A. S. AlQahtani and M. K. Khan. Comprehensive Survey: Biometric User Authentication Application, Evaluation, and Discussion. *Computers and Electrical Engineering* 119: 109485, 2024.

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. 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, volume 2. 2024.

T. Araújo, G. Aresta, E. Castro, J. Rouco, P. Aguiar *et al.* Classification of breast cancer histology images using Convolutional Neural Networks. *PLOS ONE* 12(6): 1–14, 2017.

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. Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 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.

L. Azzari and A. Foi. Iterative Poisson image denoisng software. `https://webpages.tuni.fi/foi/invansc/`, 2016a.

L. Azzari and A. Foi. Variance Stabilization for Noisy+Estimate Combination in Iterative Poisson Denoising. *IEEE Signal Processing Letters* 23(8): 1086–1090, 2016b.

O. Bachem, M. Lucic, H. Hassani and A. Krause. Fast and Provably Good Seedings for k-Means. NeurIPS, 55–63. 2016a.

O. Bachem, M. Lucic, S. H. Hassani and A. Krause. Approximate K-Means++ in Sublinear Time. Proceedings AAAI Conference on Artificial Intelligence, volume 30, 1459–1467. 2016b.

A. Baevski, W.-N. Hsu, A. Conneau and M. Auli. Unsupervised Speech Recognition. M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang and J. W. Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, 27826–27839. 2021.

D. Bahdanau, K. Cho and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473* 2014.

B. Bajić, A. Suveer, A. Gupta, I. Pepić, J. Lindblad *et al.* Denoising of short exposure transmission electron microscopy images for ultrastructural enhancement. IEEE 15th International Symposium on Biomedical Imaging, 921–925. 2018.

P. Baldi, P. Sadowski and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications* 5(1), 2014.

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. Proceedings of the 36th International Conference on Machine Learning, 524–533. 2019.

W. T. Baxter, R. A. Grassucci, H. Gao and J. Frank. Determination of signal-to-noise ratios and spectral SNRs in cryo-EM low-dose imaging of molecules. *Journal of Structural Biology* 166(2): 126–132, 2009.

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.

M. Belkin and P. Niyogi. Semi-Supervised Learning on Riemannian Manifolds. *Machine Learning* 56(1): 209–239, 2004.

T. Bepler, K. Kelley, A. J. Noble and B. Berger. Topaz-Denoise: general deep denoising models for cryoEM and cryoET. *Nature Communications* 11: 5208, 2020.

C. M. Bishop. Pattern Recognition and Machine Learning. Springer, 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.

J. Bornschein, M. Henniges and J. Lücke. Are V1 Simple Cells Optimized for Visual Occlusions? A Comparative Study. *PLOS Computational Biology* 9(6): e1003062, 2013.

N. Bouguila and W. Fan, editors. Mixture Models and Applications. Springer, 2020.

V. Boukun, J. Drefs and J. Lücke. Blind Zero-Shot Audio Restoration: A Variational Autoencoder Approach for Denoising and Inpainting. Interspeech, 4823–4827. 2024.

C. Bouveyron, S. Girard and C. Schmid. High-dimensional data clustering. *Computational Statistics & Data Analysis* 52(1): 502–519, 2007.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan *et al.* Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33: 1877–1901, 2020.

C. Bryant, Z. Yuan, M. R. Qorib, H. Cao, H. T. Ng *et al.* Grammatical Error Correction: A Survey of the State of the Art. *Computational Linguistics* 49(3): 643–701, 2023.

R. E. Bryant and D. R. O'Hallaron. Computer systems: A Programmer's Perspective. Prentice Hall, 2011.

H. C. Burger, C. J. Schuler and S. Harmeling. Image denoising: Can plain Neural Networks compete with BM3D? IEEE Conference on Computer Vision and Pattern Recognition, 2392–2399. 2012.

J. Y. K. Chan and A. P. Leung. Efficient k-means++ with Random Projection. 2017 International Joint Conference on Neural Networks (IJCNN), 94–100. 2017.

A. Chebli, A. Djebbar and H. F. Marouani. Semi-Supervised Learning for Medical Application: A Survey. 2018 International Conference on Applied Smart Systems (ICASS), 1–9. 2018.

G. Chen, F. Zhu and P. Ann Heng. An Efficient Statistical Method for Image Noise Level Estimation. IEEE International Conference on Computer Vision, 477–485. 2015a.

G. Chen, F. Zhu and P. Ann Heng. Noise Level Estimation for Signal Image. *GitHub repository.* `https://github.com/zsyOAOA/noise_est_ICCV2015`, 2015b.

X. Chen, X. Wang, K. Zhang, K.-M. Fung, T. C. Thai *et al.* Recent advances and clinical applications of deep learning in medical image analysis. *Medical Image Analysis* 79: 102444, 2022.

Y.-J. Chen, Y.-J. Chang, S.-C. Wen, Y. Shi, X. Xu *et al.* Zero-Shot Medical Image Artifact Reduction. IEEE International Symposium on Biomedical Imaging, 862–866. 2020.

D.-Y. Cheng, A. Gersho, B. Ramamurthi and Y. Shoham. Fast search algorithms for vector quantization and pattern matching. IEEE International Conference on Acoustics, Speech, and Signal Processing, volume 9, 372–375. 1984.

M. Chi, X. He and S. Yu. Mixture Model Label Propagation. Proceedings of the 19th ACM International Conference on Information and Knowledge Management, 1889–1892. 2010.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra *et al.* PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* 24(240): 1–113, 2023.

G. Cohen, S. Afshar, J. Tapson and A. van Schaik. EMNIST: Extending MNIST to handwritten letters. 2017 International Joint Conference on Neural Networks, 2921–2926. 2017.

B. Cottier, R. Rahman, L. Fattorini, N. Maslej, T. Besiroglu *et al.* The rising costs of training frontier AI models. *arXiv preprint arXiv:2405.21015* 2024.

F. Crete, T. Dolmiere, P. Ladret and M. Nicolas. The Blur Effect: Perception and Estimation with a New No-Reference Perceptual Blur Metric. B. E. Rogowitz, T. N. Pappas and S. J. Daly, editors, Human Vision and Electronic Imaging XII, volume 6492, 64920I. 2007.

G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems* 2(4): 303–314, 1989.

K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian. Image Denoising by Sparse 3D Transform-Domain Collaborative Filtering. *IEEE Transactions on Image Processing* 16(8): 2080–2095, 2007.

Z. Dai, G. Exarchakis and J. Lücke. What Are the Invariant Occlusive Components of Image Patches? A Probabilistic Generative Approach. NeurIPS, 243–251. 2013.

Z. Dai and J. Lücke. Autonomous Document Cleaning—A Generative Approach to Reconstruct Strongly Corrupted Scanned Texts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(10): 1950–1962, 2014.

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: 1–22, 1977.

L. Deng and X. Li. Machine Learning Paradigms for Speech Recognition: An Overview. *IEEE Transactions on Audio, Speech, and Language Processing* 21(5): 1060–1089, 2013.

A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester *et al.* ETA Prediction with Graph Neural Networks in Google Maps. Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 3767–3776. 2021.

C. Ding and X. He. K-means Clustering via Principal Component Analysis. Proceedings of the Twenty-First International Conference on Machine Learning, 29. 2004.

C. Dittmayer, J. Meinhardt, H. Radbruch, J. Radke, B. I. Heppner *et al.* Why misinterpretation of electron micrographs in SARS-CoV-2-infected tissue goes viral. *The Lancet* 396(10260): e64–e65, 2020.

W. Dong, P. Wang, W. Yin, G. Shi, F. Wu *et al.* Denoising Prior Driven Deep Neural Network for Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41(10): 2305–2318, 2019.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai *et al.* An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. International Conference on Learning Representations. 2021.

J. Drefs, E. Guiraud and J. Lücke. Evolutionary Variational Optimization of Generative Models. *Journal of Machine Learning Research* 23(21): 1–51, 2022.

J. Drefs, E. Guiraud, F. Panagiotou and J. Lücke. Direct Evolutionary Optimization of Variational Autoencoders with Binary Latents. Joint European Conference on Machine Learning and Knowledge Discovery in Databases. 2023.

S. Drgas, T. Virtanen, J. Lücke and A. Hurmalainen. Binary Non-Negative Matrix Deconvolution for Audio Dictionary Learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25(8): 1644–1656, 2017.

J. M. Duarte and L. Berton. A review of semi-supervised learning for text classification. *Artificial Intelligence Review* 56(9): 9401–9469, 2023.

J. M. Ede and R. Beanland. Partial Scanning Transmission Electron Microscopy with Deep Learning. *Scientific Reports* 10(1): 8332, 2020.

R. R. Eguchi, C. A. Choe and P.-S. Huang. Ig-VAE: Generative modeling of protein structure by direct 3D coordinate generation. *PLOS Computational Biology* 18(6): 1–18, 2022.

M. Elad and M. Aharon. Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries. *IEEE Transactions on Image Processing* 15(12): 3736–3745, 2006.

G. Elidan and N. Friedman. The Information Bottleneck EM Algorithm. *arXiv preprint arXiv:1212.2460* 2012.

C. Elkan. Using the Triangle Inequality to Accelerate k-Means. ICML, volume 3, 147–153. 2003.

M. Emad, M. Peemen and H. Corporaal. DualSR: Zero-Shot Dual Learning for Real-World Super-Resolution. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 1630–1639. 2021.

D. Eppstein, M. S. Paterson and F. F. Yao. On Nearest-Neighbor Graphs. *Discrete & Computational Geometry* 17(3): 263–282, 1997.

M. D. Escobar and M. West. Bayesian Density Estimation and Inference Using Mixtures. *Journal of the American Statistical Association* 90(430): 577–588, 1995.

EVO developers. EVO - Evolutionary Variational Optimization of Generative Models. *GitHub repository.* `https://github.com/tvlearn/evo`, 2022.

G. Exarchakis, O. Oubari and G. Lenz. A Sampling-Based Approach for Efficient Clustering in Large Datasets. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 12403–12412. 2022.

L. Fan, F. Zhang, H. Fan and C. Zhang. Brief review of image denoising techniques. *Visual Computing for Industry, Biomedicine, and Art* 2(1): 7, 2019.

Y. Fang, W. Wang, B. Xie, Q. Sun, L. Wu *et al.* EVA: Exploring the Limits of Masked Visual Representation Learning at Scale. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 19358–19369. 2023.

D. Feldman, M. Faulkner and A. Krause. Scalable Training of Mixture Models via Coresets. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira and K. Weinberger, editors, Advances in Neural Information Processing Systems, volume 24, 2142–2150. 2011.

W. Feng, P. Qiao and Y. Chen. Fast and Accurate Poisson Denoising With Trainable Nonlinear Diffusion. *IEEE Transactions on Cybernetics* 48(6): 1708–1719, 2018.

D. Forster and J. Lücke. Can clustering scale sublinearly with its clusters? A variational EM acceleration of GMMs and k-means. A. Storkey and F. Perez-Cruz, editors, Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, volume 84, 124–132. 2018.

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.

A. Ganguly, S. Jain and U. Watchareeruetai. Amortized Variational Inference: A Systematic Review. *Journal of Artificial Intelligence Research* 78: 167–215, 2023.

Z. Ghahramani and G. E. Hinton. The EM Algorithm for Mixtures of Factor Analyzers. *Technical report*, University of Toronto, 1996.

B. Ghojogh, M. Crowley, F. Karray and A. Ghodsi. Laplacian-Based Dimensionality Reduction, 249–284. Springer International Publishing, Cham, 2023.

I. Goodfellow, A. C. Courville and Y. Bengio. Large-Scale Feature Learning With Spike-and-Slab Sparse Coding. International Conference on Machine Learning. 2012.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley *et al.* Generative Adversarial Nets. *Advances in Neural Information Processing Systems* 27, 2014.

S. Gu, L. Zhang, W. Zuo and X. Feng. Weighted Nuclear Norm Minimization with Application to Image Denoising. IEEE Conference on Computer Vision and Pattern Recognition, 2862–2869. 2014.

G. Guennebaud, B. Jacob *et al.* Eigen v3. `http://eigen.tuxfamily.org`, 2010.

S. Haider, A. Cameron, P. Siva, D. Lui, M. Shafiee *et al.* Fluorescence microscopy image noise reduction using a stochastically-connected random field model. *Scientific Reports* 6(1): 20640, 2016.

S. Har-Peled and S. Mazumdar. On Coresets for k-Means and k-Median Clustering. Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 291–300. 2004.

C. R. Harris, K. J. Millman, S. J. van der Walt *et al.* Array programming with NumPy. *Nature* 585(7825): 357–362, 2020.

K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.

D. Hellwig, N. C. Hellwig, S. Boehner, T. Fuchs, R. Fischer *et al.* Artificial Intelligence and Deep Learning for Advancing PET Image Reconstruction: State-of-the-Art and Future Directions. *Nuklearmedizin* 62(06): 334–342, 2023.

T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse *et al.* Scaling Laws for Autoregressive Generative Modeling. *arXiv preprint 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.

G. E. Hinton and T. J. Sejnowski. Learning and Relearning in Boltzmann Machines, 282–317. MIT Press, Cambridge, MA, USA, 1986.

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.

J. Ho, A. Jain and P. Abbeel. Denoising Diffusion Probabilistic Models. *Advances in Neural Information Processing Systems* 33: 6840–6851, 2020.

M. D. Hoffman, D. M. Blei, C. Wang and J. Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research* 14(1): 1303–1347, 2013.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. volume 79, 2554–2558. 1982.

K. Hornik, M. Stinchcombe and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* 2(5): 359–366, 1989.

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.

T. Huang, S. Li, X. Jia, H. Lu and J. Liu. Neighbor2Neighbor: Self-Supervised Denoising from Single Noisy Images. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 14781–14790. 2021.

M. C. Hughes and E. B. Sudderth. Fast Learning of Clusters and Topics via Sparse Posteriors. *arXiv preprint arXiv:1609.07521* 2016.

J. Hull. A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(5): 550–554, 1994.

M. Hüpfel, A. Y. Kobitski, W. Zhang and G. U. Nienhaus. Wavelet-based background and noise subtraction for fluorescence microscopy images. *Biomedical Optics Express* 12(2): 969–980, 2021.

S. Hurault, T. Ehret and P. Arias. EPLL: An Image Denoising Method Using a Gaussian Mixture Model Learned on a Large Set of Patches. *Image Processing On Line* 8: 465–489, 2018.

A. Hyvärinen. A unified probabilistic model for independent and principal component analysis. E. Bingham, S. Kaski, J. Laaksonen and J. Lampinen, editors, Advances in Independent Component Analysis and Learning Machines, 75–82. 2015.

International Organization of Standardization (ISO). Standard C++. `https://isocpp.org/`, 2025.

Z. Irace and H. Batatia. Bayesian spatiotemporal segmentation of combined PET-CT data using a bivariate Poisson mixture model. European Signal Processing Conference, 2095–2099. 2014.

L. Ivanova, A. Buch, K. Döhner, A. Pohlmann, A. Binz *et al.* Conserved Tryptophan Motifs in the Large Tegument Protein pUL36 Are Required for Efficient Secondary Envelopment of Herpes Simplex Virus Capsids. *Journal of Virology* 90(11): 5368–5383, 2016.

W. Jakob, J. Rhinelander and D. Moldovan. pybind11 – Seamless operability between C++11 and Python. `https://github.com/pybind/pybind11`, 2017.

X. Jia, Y. Tong, H. Qiao, M. Li, J. Tong *et al.* Fast and accurate object detector for autonomous driving based on improved YOLOv5. *Scientific Reports* 13(1): 9711, 2023.

M. Jordan, Z. Ghahramani, T. Jaakkola and L. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning* 37: 183–233, 1999.

M. I. Jordan and R. A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation* 6(2): 181–214, 1994.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess *et al.* Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361* 2020.

M. E. Khan, G. Bouchard, B. M. Marlin and K. P. Murphy. Variational Bounds for Mixed-Data Factor Analysis. Proceedings of the 24th International Conference on Neural Information Processing Systems, 1108–1116. 2010.

J. Kim, D. Lee, H. Lim, H. Yang, J. Kim *et al.* Deep learning alignment of bidirectional raster scanning in high speed photoacoustic microscopy. *Scientific Reports* 12(1): 16238, 2022.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* 2014.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. International Conference on Learning Representations (ICLR). 2014.

T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations. 2017.

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. Krizhevsky, V. Nair and G. Hinton. The CIFAR-10 dataset. *online: https://www.cs.toronto.edu/ kriz/cifar.html* 2014.

A. Krizhevsky, I. Sutskever and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 25, 2012.

A. Krull, T.-O. Buchholz and F. Jug. Noise2Void - Learning Denoising from Single Noisy Images. IEEE Conference on Computer Vision and Pattern Recognition, 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.

A. Krull, T. Vicar, M. Prakash, M. Lalit and F. Jug. Convallaria dataset for microscopy image denoising benchmark as used in Probabilistic Noise2Void paper. 2020a.

A. Krull, T. Vičar, M. Prakash, M. Lalit and F. Jug. Probabilistic Noise2Void: Unsupervised Content-Aware Denoising. *Frontiers in Computer Science* 2: 5, 2020b.

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.

M. M. Lamers, J. Beumer, J. van der Vaart, K. Knoops, J. Puschhof *et al.* SARS-CoV-2 productively infects human gut enterocytes. *Science* 369(6499): 50–54, 2020.

H. Larochelle and I. Murray. The Neural Autoregressive Distribution Estimator. Proceedings of the fourteenth international conference on artificial intelligence and statistics, 29–37. 2011.

M. Laue, A. Kauter, T. Hoffmann, J. Michel and A. Nitsche. Electron microscopy of SARS-CoV-2 particles - Dataset 02. 2020a.

M. Laue, A. Kauter, T. Hoffmann, J. Michel and A. Nitsche. Electron microscopy of SARS-CoV-2 particles - Dataset 03. 2020b.

M. Laue, A. Kauter, T. Hoffmann, J. Michel and A. Nitsche. Electron microscopy of SARS-CoV-2 particles - Dataset 07. 2020c.

M. Laue, A. Kauter, T. Hoffmann, L. Möller, J. Michel *et al.* Morphometry of SARS-CoV and SARS-CoV-2 particles in ultrathin plastic sections of infected Vero cell cultures. *Scientific Reports* 11(1): 3515, 2021.

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* 86(11): 2278–2324, 1998.

A. B. Lee, K. S. Pedersen and D. Mumford. The Nonlinear Statistics of High-Contrast Patches in Natural Images. *International Journal of Computer Vision* 54: 83–103, 2003.

Z. Lee, H. Rose, O. Lehtinen, J. Biskupek and U. Kaiser. Electron dose dependence of signal-to-noise ratio, atom contrast and resolution in transmission electron microscope images. *Ultramicroscopy* 145: 3–12, 2014.

J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras *et al.* Noise2Noise: Learning Image Restoration without Clean Data. International Conference on Machine Learning, 2965–2974. 2018.

J. Lequyer, R. Philip, A. Sharma, W.-H. Hsu and L. Pelletier. A fast blind zero-shot denoiser. *GitHub repository.* `https://github.com/pelletierlab/Noise2Fast`, 2022a.

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, 2022b.

J. Q. Li and A. R. Barron. Mixture Density Estimation. Proceedings of the 12th International Conference on Neural Information Processing Systems, 279–285. 1999.

Z. Li, C. Li, L. Yang, P. S. Yu and Z. Li. Mixture distribution modeling for scalable graph-based semi-supervised learning. *Knowledge-Based Systems* 200: 105974, 2020.

W. Lin, M. E. Khan and M. Schmidt. Fast and Simple Natural-Gradient Variational Inference with Mixture of Exponential-family Approximations. International Conference on Machine Learning, 3992–4002. 2019.

J. Liu, R. Jiang, X. Liu, F. Zhou, Y. Chen *et al.* Large-Scale Clustering on 100 M-Scale Datasets Using a Single T4 GPU via Recall KNN and Subgraph Segmentation. *Neural Processing Letters* 56(1): 34, 2024.

W. Liu, J. He and S.-F. Chang. Large Graph Construction for Scalable Semi-Supervised Learning. Proceedings of the 27th International Conference on International Conference on Machine Learning, 679–686. 2010.

Z. Liu, P. Luo, X. Wang and X. Tang. Deep Learning Face Attributes in the Wild. Proceedings of the IEEE International Conference on Computer Vision, 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.

G. Loosli, S. Canu and L. Bottou. Training Invariant Support Vector Machines using Selective Sampling. L. Bottou, O. Chapelle, D. DeCoste and J. Weston, editors, Large Scale Kernel Machines, 301–320. 2007.

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: 2855–2900, 2010.

J. Lücke and D. Forster. k-means as a variational EM approximation of Gaussian mixture models. *Pattern Recognition Letters* 125: 349–356, 2019.

J. Lücke and M. Sahani. Maximal Causes for Non-linear Component Extraction. *Journal of Machine Learning Research* 9(41): 1227–1267, 2008.

J. Lücke, R. Turner, M. Sahani and M. Henniges. Occlusive Components Analysis. Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams and A. Culotta, editors, Advances in Neural Information Processing Systems, volume 22. 2009.

J. Mairal, M. Elad and G. Sapiro. Sparse Representation for Color Image Restoration. *IEEE Transactions on Image Processing* 17(1): 53–69, 2008.

Y. Mäkinen. Python wrapper for BM3D denoising. *Python Package Index*. `https://pypi.org/project/bm3d`, 2007.

M. Mäkitalo and A. Foi. Denoising software for Poisson and Poisson-Gaussian data. `https://webpages.tuni.fi/foi/invansc/`, 2011.

M. Makitalo and A. Foi. Optimal Inversion of the Anscombe Transformation in Low-Count Poisson Image Denoising. *IEEE Transactions on Image Processing* 20(1): 99–109, 2011.

L. Manduchi, K. Pandey, C. Meister, R. Bamler, R. Cotterell *et al.* On the Challenges and Opportunities in Generative AI. *arXiv preprint arXiv:2403.00025* 2024.

K. Marstal, F. Berendsen, M. Staring and S. Klein. SimpleElastix: A User-Friendly, Multilingual Library for Medical Image Registration. 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 574–582. 2016.

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. Proceedings Eighth IEEE International Conference on Computer Vision, volume 2, 416–423. 2001.

N. Maslej, L. Fattorini, R. Perrault, V. Parli, A. Reuel *et al.* The AI Index 2024 Annual Report. AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, 2024. `https://aiindex.stanford.edu/report/`.

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. Wiley, 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.

Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 4.1, 2023.

S. Mohamed, K. A. Heller and Z. Ghahramani. Bayesian and L1 Approaches for Sparse Unsupervised Learning. Proceedings of the 29th International Coference on International Conference on Machine Learning, 683–690. 2012.

T. Monk, C. Savin and J. Lücke. Optimal neural inference of stimulus intensities. *Scientific Reports* 8(1): 10038, 2018.

H. Mousavi, M. Buhl, E. Guiraud, J. Drefs and J. Lücke. Inference and Learning in a Latent Variable Model for Beta Distributed Interval Data. *Entropy* 23(5): 552, 2021.

H. Mousavi, J. Drefs, F. Hirschberger and J. Lücke. Generic Unsupervised Optimization for a Latent Variable Model With Exponential Family Observables. *Journal of Machine Learning Research* 24(285): 1–59, 2023.

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.

K. P. Murphy. Machine Learning: A Probabilistic Perspective. MIT press, 2012.

K. P. Murphy. Probabilistic Machine Learning: An Introduction. MIT Press, 2022.

P. Ndajah, H. Kikuchi, M. Yukawa, H. Watanabe and S. Muramatsu. SSIM Image Quality Metric for Denoised Images. 3rd WSEAS International Conference on Visualization, Imaging and Simulation, 53–57. 2010.

R. Neal and G. Hinton. A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants. Learning in Graphical Models, 355–368. 1998.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu *et al.* Reading Digits in Natural Images with Unsupervised Feature Learning. NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. 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.

J. Nickolls, I. Buck, M. Garland and K. Skadron. Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue* 6(2): 40–3, 2008.

F. Nie, D. Xu, I. W.-H. Tsang and C. Zhang. Flexible Manifold Embedding: A Framework for Semi-Supervised and Unsupervised Dimension Reduction. *IEEE Transactions on Image Processing* 19(7): 1921–1932, 2010.

Nobel Prize. The Nobel Prize in Physics 2024. `https://www.nobelprize.org/prizes/physics/2024/summary/`, 2024. Accessed: 2025-06-30.

A. Oliver, A. Odena, C. Raffel, E. D. Cubuk and I. J. Goodfellow. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. Proceedings of the 32nd International Conference on Neural Information Processing Systems, 3239–3250. 2018.

B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* 381: 607–9, 1996.

OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.5. 2015.

M. Opper, O. Winther and M. J. Jordan. Expectation Consistent Approximate Inference. *Journal of Machine Learning Research* 6(12), 2005.

N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics* 9(1): 62–66, 1979.

V. Papyan and M. Elad. Multi-Scale Patch-Based Image Restoration. *IEEE Transactions on Image Processing* 25(1): 249–261, 2016.

S. Parameswaran, C.-A. Deledalle, L. Denis and T. Q. Nguyen. Accelerating GMM-Based Patch Priors for Image Restoration: Three Ingredients for a 100× Speed-Up. *IEEE Transactions on Image Processing* 28(2): 687–698, 2018.

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.

G. Peyré. Manifold models for signals and images. *Computer vision and image understanding* 113(2): 249–260, 2009.

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. International Conference on Learning Representations. 2021b.

M. Prakash, M. Lalit, P. Tomancak, A. Krull and F. Jug. Mouse actin dataset for microscopy image denoising benchmark as used in PPN2V paper. 2019a.

M. Prakash, M. Lalit, P. Tomancak, A. Krull and F. Jug. Mouse skull nuclei dataset for microscopy image denoising benchmark as used in PPN2V paper. 2019b.

M. Prakash, M. Lalit, P. Tomancak, A. Krull and F. Jug. Fully Unsupervised Probabilistic Noise2Void. IEEE International Symposium on Biomedical Imaging, 154–158. 2020.

S. Qiu, F. Nie, X. Xu, C. Qing and D. Xu. Accelerating Flexible Manifold Embedding for Scalable Semi-Supervised Learning. *IEEE Transactions on Circuits and Systems for Video Technology* 29(9): 2786–2795, 2019.

Y. Quan, M. Chen, T. Pang and H. Ji. Self2Self With Dropout: Learning Self-Supervised Denoising From Single Image. IEEE Conference on Computer Vision and Pattern Recognition, 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.

M. Rattray, O. Stegle, K. Sharp and J. Winn. Inference algorithms and learning theory for Bayesian sparse factor analysis. Journal of Physics: Conference Series, volume 197. 2009.

J. Redmon, S. Divvala, R. Girshick and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 779–788. 2016.

T. Remez, O. Litany, R. Giryes and A. M. Bronstein. Deep Convolutional Denoising of Low-Light Images. *arXiv preprint arXiv:1701.01687* 2017.

D. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. International Conference on Machine Learning, 1530–1538. 2015.

D. J. Rezende, S. Mohamed and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. International Conference on Machine Learning (ICML). 2014.

E. Richardson. torch-mfa. *GitHub Repository.* `https://github.com/eitanrich/torch-mfa`, 2019.

E. Richardson and Y. Weiss. On GANs and GMMs. Proceedings of the 32nd International Conference on Neural Information Processing Systems, 5852–5863. 2018.

S. Robin and L. Scrucca. Mixture-based estimation of entropy. *Computational Statistics & Data Analysis* 177: 107582, 2023.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10684–10695. 2022.

O. Ronneberger, P. Fischer and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention, 234–241. 2015.

C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang *et al.* Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *Advances in Neural Information Processing Systems* 35: 36479–36494, 2022.

S. Salwig. Denoising with Poisson Mixture Models. *GitHub repository.* `https://github.com/salwig/pmm`, 2024.

S. Salwig, J. Drefs and J. Lücke. Short-Exposure Transmission Electron Microscopy of Cilia. 2024.

S. Salwig, T. Kahlke, F. Hirschberger, D. Forster and J. Lücke. Sublinear Variational Optimization of Gaussian Mixture Models with Millions to Billions of Parameters. *arXiv preprint arXiv:2501.12299* 2025.

L. K. Saul and M. I. Jordan. Exploiting Tractable Substructures in Intractable Networks. Advances in Neural Information Processing Systems, volume 8. 1995.

D. Sculley. Web-Scale K-Means Clustering. Proceedings of the 19th International Conference on World Wide Web, 1177–1178. 2010.

M. Shehab, L. Abualigah, Q. Shambour, M. A. Abu-Hashem, M. K. Y. Shambour *et al.* Machine learning in medical applications: A review of state-of-the-art methods. *Computers in Biology and Medicine* 145: 105458, 2022.

A.-S. Sheikh, N. S. Harper, J. Drefs, Y. Singer, Z. Dai *et al.* STRFs in primary auditory cortex emerge from masking-based statistics of natural sounds. *PLOS Computational Biology* 15(1): 1–23, 2019.

A.-S. Sheikh, J. A. Shelton and J. Lücke. A Truncated EM Approach for Spike-and-Slab Sparse Coding. *Journal of Machine Learning Research* 15: 2653–2687, 2014.

J. Shelton, A. Sheikh, P. Berkes, J. Bornschein and J. Lücke. Select and Sample - A Model of Efficient Neural Inference and Learning. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira and K. Weinberger, editors, Advances in Neural Information Processing Systems, volume 24. 2011.

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.

J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. *Technical report*, Carnegie Mellon University, 1994.

A. Shocher, N. Cohen and M. Irani. Zero-Shot Super-Resolution Using Deep Internal Learning. IEEE Conference on Computer Vision and Pattern Recognition, 3118–3126. 2018.

T. Sieberth, R. Wackrow and J. H. Chandler. Automatic Detection of Blurred Images in UAV Image Sets. *ISPRS Journal of Photogrammetry and Remote Sensing* 122: 1–16, 2016.

D. Slate. Letter Recognition. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5ZP40.

J. W. Soh, S. Cho and N. I. Cho. Meta-Transfer Learning for Zero-Shot Super-Resolution. IEEE Conference on Computer Vision and Pattern Recognition, 3513–3522. 2020.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan and S. Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. International Conference on Machine Learning, 2256–2265. 2015.

Y. Song, J. Zhang and C. Zhang. A survey of large-scale graph-based semi-supervised classification algorithms. *International Journal of Cognitive Computing in Engineering* 3: 188–198, 2022.

Z. Song, X. Yang, Z. Xu and I. King. Graph-Based Semi-Supervised Learning: A Comprehensive Review. *IEEE Transactions on Neural Networks and Learning Systems* 34(11): 8174–8194, 2023.

A. B. Szczotka, D. I. Shakir, M. J. Clarkson, S. P. Pereira and T. Vercauteren. Zero-Shot Super-Resolution With a Physically-Motivated Downsampling Kernel for Endomicroscopy. *IEEE Transactions on Medical Imaging* 40(7): 1863–1874, 2021.

Y. Tai, J. Yang, X. Liu and C. Xu. MemNet: A Persistent Memory Network for Image Restoration. IEEE International Conference on Computer Vision, 4539–4547. 2017.

O. Tange. GNU Parallel 20210822 ('Kabul'). 2021.

The Boost organization. Boost C++ Libraries. `http://www.boost.org`, 2024.

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.

C. Tian, Y. Xu and W. Zuo. Image denoising using deep CNN with batch renormalization. *Neural Networks* 121: 461–473, 2020.

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.

M. E. Tipping and C. M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 61(3): 611–622, 1999.

M. K. Titsias and M. Lázaro-Gredilla. Spike and Slab Variational Inference for Multi-Task and Multiple Kernel Learning. Advances in Neural Information Processing Systems, volume 24. 2011.

D. Ulyanov, A. Vedaldi and V. Lempitsky. Deep Image Prior. IEEE Conference on Computer Vision and Pattern Recognition, 9446–9454. 2018.

A. Van den Oord and B. Schrauwen. Factoring Variations in Natural Images with Deep Gaussian Mixture Models. *Advances in Neural Information Processing Systems* 27, 2014.

J. E. van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine Learning* 109(2): 373–440, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones *et al.* Attention is All you Need. *Advances in Neural Information Processing Systems* 30, 2017.

C. Viroli and G. J. McLachlan. Deep Gaussian Mixture Models. *Statistics and Computing* 29: 43–51, 2019.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17: 261–272, 2020.

C. Vonesch, F. Aguet, J.-L. Vonesch and M. Unser. The Colored Revolution of Bioimaging. *IEEE Signal Processing Magazine* 23(3): 20–31, 2006.

F. Wang, T. R. Henninen, D. Keller and R. Erni. Noise2Atom: unsupervised denoising for scanning transmission electron microscopy images. *Applied Microscopy* 50: 23, 2020.

M. Wang, W. Fu, S. Hao, H. Liu and X. Wu. Learning on Big Graph: Label Inference and Regularization with Anchor Hierarchy. *IEEE Transactions on Knowledge and Data Engineering* 29(5): 1101 – 1114, 2017.

M. Wang, W. Fu, S. Hao, D. Tao and X. Wu. Scalable Semi-Supervised Learning by Efficient Anchor Graph Regularization. *IEEE Transactions on Knowledge and Data Engineering* 28(7): 1864–1877, 2016.

Z. Wang and A. C. Bovik. Mean Squared Error: Love It or Leave It? A new look at signal fidelity measures. *IEEE Signal Processing Magazine* 26(1): 98–117, 2009.

Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13(4): 600–612, 2004.

Z. Wei, J. Wang, H. Nichol, S. Wiebe and D. Chapman. A median-Gaussian filtering framework for Moiré pattern noise removal from X-ray microscopy image. *Micron* 43(2-3): 170–176, 2012.

B. Workshop, T. L. Scao, A. Fan, C. Akiki, E. Pavlick *et al.* BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv preprint arXiv:2211.05100* 2022.

M. Wortsman, T. Dettmers, L. Zettlemoyer, A. Morcos, A. Farhadi *et al.* Stable and low-precision training for large-scale vision-language models. Proceedings of the 37th International Conference on Neural Information Processing Systems, 10271 – 10298. 2024.

Q. Wu, F. Merchant and K. Castleman. Microscope Image Processing. Elsevier Science, 2008.

Y. Xiang, L. Shi, J. L. Højvang, M. H. Rasmussen and M. G. Christensen. A Novel NMF-HMM Speech Enhancement Algorithm Based on Poisson Mixture Model. IEEE International Conference on Acoustics, Speech and Signal Processing, 721–725. 2021.

S. Yang and B.-U. Lee. Poisson-Gaussian Noise Reduction Using the Hidden Markov Model in Contourlet Domain for Fluorescence Microscopy Images. *PLOS ONE* 10(9): 1–19, 2015.

R. Yoshida and M. West. Bayesian Learning in Sparse Graphical Factor Models via Variational Mean-Field Annealing. *Journal of Machine Learning Research* 11: 1771–1798, 2010.

L. Yu, J. Luo, S. Xu, X. Chen and N. Xiao. An Unsupervised Weight Map Generative Network for Pixel-Level Combination of Image Denoisers. *Applied Sciences* 12(12), 2022.

A. J. Zeevi and R. Meir. Density Estimation Through Convex Combinations of Densities: Approximation and Estimation Bounds. *Neural Networks* 10(1): 99–109, 1997.

C. Zhang, J. Bütepage, H. Kjellström and S. Mandt. Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41(8): 2008–2026, 2019.

J. Zhang and K. Hirakawa. Improved Denoising via Poisson Mixture Modeling of Image Sensor Noise. *IEEE Transactions on Image Processing* 26(4): 1565–1578, 2017.

K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing* 26(7): 3142–3155, 2017.

K. Zhang, W. Zuo and L. Zhang. FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising. *IEEE Transactions on Image Processing* 2018.

Y. Zhang, S. Ji, C. Zou, X. Zhao, S. Ying *et al.* Graph Learning on Millions of Data in Seconds: Label Propagation Acceleration on Graph Using Data Distribution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45(2): 1835–1847, 2023.

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.

D. Zhou, O. Bousquet, T. N. Lal, J. Weston and B. Schölkopf. Learning with Local and Global Consistency. Proceedings of the 17th International Conference on Neural Information Processing Systems, 321–328. 2003.

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.

M. Zhou, H. Chen, J. Paisley, L. Ren, L. Li *et al.* Nonparametric Bayesian Dictionary Learning for Analysis of Noisy and Incomplete Images. *IEEE Transactions on Image Processing* 21(1): 130–144, 2012.

M. Zhou, H. Chen, L. Ren, G. Sapiro, L. Carin *et al.* Non-Parametric Bayesian Dictionary Learning for Sparse Image Representations. *Advances in Neural Information Processing Systems* 22, 2009.

X. Zhu and Z. Ghahramani. Learning from Labeled and Unlabeled Data with Label Propagation. *Technical Report CMU-CALD-02-107*, Carnegie Mellon University, 2002.

X. Zhu, Z. Ghahramani and J. Lafferty. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. Proceedings of the Twentieth International Conference on International Conference on Machine Learning, 912–919. 2003.

X. Zhu and J. Lafferty. Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. Proceedings of the 22nd International Conference on Machine Learning, 1052–1059. 2005.

D. Zoran and Y. Weiss. From Learning Models of Natural Image Patches to Whole Image Restoration. IEEE International Conference on Computer Vision, 479–486. 2011.