universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Deep Clustering of Tabular Data Using Diffusion Models

verfasst von | submitted by

Botond Jenő Kovács

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von | Supervisor:	Assoz. Prof. Dipl.-Ing. Dr.techn. Sebastian Tschiatschek BSc

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Sebastian Tschiatschek, for his invaluable patience, guidance, and thoughtful feedback throughout the development of this thesis.

I am incredibly grateful to my family, especially my mother, sister, and brother, who stood by me when I decided to switch my Master's programme and move to Austria. Their constant emotional support and encouragement helped me through every step of the process.

Lastly, I am profoundly thankful to my peers, Predrag Djindjic, Thomas Geier, Thalis Goldschmidt, Matej Vedak, and Felix Wille, whose friendship, insight, and inspiration enriched both my academic and personal experience. I could not have undertaken this journey without them.

Abstract

Deep clustering, an approach in which neural networks learn clustering-friendly latent representations that are subsequently used for clustering, has gained increasing attention in recent years. While substantial progress has been made on deep clustering of image and text datasets, tabular data remain heavily underexplored despite their importance in scientific, industrial, and financial applications. The lack of spatial structure and the mix of categorical and continuous features in tabular datasets create challenges for both representation quality and the effective application of neural networks.

At the same time, in a different line of research, namely generative modeling, a powerful class of models, diffusion models, emerged, consistently surpassing the performance of previous approaches. Although diffusion models were initially designed for generating images, they have since been adapted for both deep clustering and tabular data generation in separate tasks. This naturally motivates the question: Could these directions be merged into a unified diffusion-based algorithm for deep clustering on tabular data?

This work addresses this question by developing **CluTaD**, a deep clustering algorithm that integrates a tabular data-oriented diffusion model and a tabular data-oriented objective function. The proposed algorithm maps the original input features into latent representations, which are then clustered using a Gaussian Mixture Model. During the training phase, first noise is added to the input data, then a diffusion model tailored for tabular data is used for denoising, conditioned on the aforementioned latent features. This objective incentivizes the latent representations to retain maximal information from the original data. Simultaneously, the clustering assignments are optimized for distributional sharpness via a multilayer perceptron. The joint optimization of the denoising and the clustering objectives ensures that the latent representations are tailored for cluster separability and robust enough to reconstruct the original data characteristics.

The method was evaluated against 12 baseline clustering algorithms across 16 tabular datasets, covering different dimensionalities, cluster structures, and feature types. The results indicate that **CluTaD** falls short in settings with very high-dimensional feature spaces or in heavily imbalanced scenarios with only two clusters. Specifically, on the dataset with 70 features, the model achieved a clustering accuracy of only 54.7%, even though existing approaches reach as high as 99.9%. Furthermore, on imbalanced bi-cluster datasets, it consistently collapsed into a single-cluster solution. However, the model demonstrated robust performance across the remaining datasets. Specifically, it achieved the highest clustering accuracy in 8 out of the 16 datasets. The most notable improvement was a 5.3% increase in accuracy over the previous state-of-the-art results.

Overall, this thesis introduces a deep clustering algorithm tailored for tabular data that utilizes a diffusion model architecture. Experimental comparisons on benchmark datasets suggest that while limitations exist, the proposed approach demonstrates state-of-the-art performance across most of the evaluated scenarios.

Abstract

Deep Clustering, ein Ansatz, bei dem neuronale Netze clustering-optimierte latente Repräsentationen erlernen, hat in den letzten Jahren an Bedeutung gewonnen. Während Deep Clustering für Bild- und Textdaten fortgeschritten ist, bleiben tabellarische Daten trotz ihrer wissenschaftlichen und wirtschaftlichen Relevanz untererforscht. Die fehlenden räumlichen Strukturen und Mischung aus Merkmalstypen erschweren hierbei die Repräsentationsqualität sowie die Anwendung neuronaler Netze.

Gleichzeitig etablierten sich Diffusionsmodelle als leistungsstarke generative Modelle, die bisherige Ansätze konsequent übertreffen. Obwohl ursprünglich für Bilder entwickelt, wurden sie bereits für Deep Clustering und tabellarische Datengenerierung adaptiert. Dies führt zur Frage: Können Diffusionsmodelle einen wettbewerbsfähigen Deep-Clustering-Algorithmus für tabellarische Daten ermöglichen?

Diese Arbeit adressiert diese Frage durch die Entwicklung von **CluTaD**, einem Deep-Clustering-Algorithmus, der ein auf tabellarische Daten ausgerichtetes Diffusionsmodell und eine auf tabellarische Daten ausgerichtete Zielfunktion integriert. Der vorgeschlagene Algorithmus bildet die ursprünglichen Eingangsmerkmale in latente Repräsentationen ab, die anschließend mittels eines Gaußschen Mischmodells (Gaussian Mixture Model) geclustert werden. Während der Trainingsphase wird den Eingangsdaten zuerst Rauschen hinzugefügt, danach dient ein speziell für tabellarische Daten entwickeltes Diffusionsmodell dem Denoising, konditioniert auf die oben genannten latenten Features. Dieses Ziel incentiviert die latenten Repräsentationen, ein Maximum an Information aus den Originaldaten beizubehalten. Gleichzeitig werden die Cluster-Zuweisungen mittels eines Multi-Layer-Perzeptrons auf Verteilungsschärfe optimiert. Die gemeinsame Optimierung der Entrauschungs- und Clustering-Ziele stellt sicher, dass die latenten Repräsentationen auf die Cluster-Trennbarkeit zugeschnitten und robust genug sind, um die ursprünglichen Datencharakteristika zu rekonstruieren.

Die Methode wurde im Vergleich zu 12 Baseline-Clustering-Algorithmen anhand von 16 tabellarischen Datensätzen evaluiert, die verschiedene Dimensionalitäten, Clusterstrukturen und Merkmalsarten abdecken. Die Ergebnisse deuten darauf hin, dass **CluTaD** in Umgebungen mit sehr hochdimensionalen Merkmalsräumen oder in stark unbalancierten Szenarien mit nur zwei Clustern hinter den Erwartungen zurückbleibt. Speziell bei dem Datensatz mit 70 Merkmalen erreichte das Modell eine Clustering-Genauigkeit von nur 54.7%, obwohl bestehende Ansätze bis zu 99.9% erreichen. Zudem kollabierte es bei den unbalancierten Bi-Cluster-Datensätzen konsistent zu einer Ein-Cluster-Lösung. Dennoch demonstrierte das Modell eine robuste Leistung bei den übrigen Datensätzen. Konkret erzielte es bei 8 der 16 Datensätze die höchste Clustering-Genauigkeit. Die nennenswerteste Verbesserung war eine Steigerung der Genauigkeit um 5.3% gegenüber den bisherigen State-of-the-Art-Ergebnissen.

Insgesamt führt diese Thesis einen diffusionsbasierten Deep-Clustering-Algorithmus für tabellarische Daten ein. Experimentelle Vergleiche zeigen, dass der Ansatz trotz spezifischer Limitationen in den meisten Szenarien State-of-the-Art-Leistungen erbringt.

Contents

List of Tables

List of Figures

List of Algorithms

1. Introduction

Unsupervised learning refers to a set of methods that aim to extract meaningful structure from unlabeled data. It includes tasks such as clustering, dimensionality reduction, and anomaly detection, all of which seek to reveal hidden patterns or relationships within the data. Clustering is a central part of this field, used to group similar data points based on their features.

A major challenge for traditional clustering algorithms arises when the original features do not clearly separate the underlying groups. While performing feature transformation and clustering the resulting representations can address this in some scenarios, these approaches often struggle to capture complex structures. More importantly, most feature transformation techniques do not take into account that the end goal is clustering.

These issues can be mitigated by employing neural networks (NNs) for non-linear feature transformations specifically for clustering. Deep neural networks offer high flexibility in modeling complex patterns, and their training can be guided directly by clustering objectives [1]. This led to the birth of the field of deep clustering (DC), which applies NNs in clustering tasks. Foundational approaches in DC include Deep Embedded Clustering (DEC) [2], Deep Clustering Network (DCN) [3], and Improved Deep Embedded Clustering (IDEC) [4]. These models have become standard benchmarks and are widely used for evaluating novel clustering methods, however, mainly for image and text datasets.

In contrast to tabular data, these dataset types have homogeneous, continuous, and high-dimensional features with a spatial structure that aligns with the underlying assumptions of most deep clustering architectures [5]. While images and text can always be represented as vectors, they still possess a structure that DC algorithms are designed to exploit. Throughout this thesis, the term tabular data refers exclusively to classical tabular data (e.g., spreadsheets or databases) and excludes other data types, like images or natural language.

Consequently, although deep learning has seen remarkable progress in vision and language tasks, it has traditionally struggled to match the performance of classical machine learning methods on tabular datasets [6], including deep clustering methods [7]. This remains an open challenge, particularly in domains such as finance, healthcare, and logistics, where tabular data is ubiquitous. Nevertheless, recent advances such as G-CEALS [5], a deep clustering framework specifically designed for tabular data, have demonstrated superior performance compared to previous clustering approaches.

Many deep clustering frameworks leverage generative models to perform the feature transformation step. Variational autoencoders (VAEs) and generative adversarial networks (GANs) [8] are among the most commonly used frameworks when selecting the NN architecture to use for feature transformation. More recently, denoising diffusion probabilistic models (DDPMs) [9] have emerged as powerful generative models, achieving

state-of-the-art performance in image synthesis. DDPMs operate by reversing a fixed noising process, iteratively transforming Gaussian noise into realistic samples.

Recent adaptations [10], including a model called ClusterDDPM [11], have even extended DDPMs for unsupervised learning tasks. This demonstrates that, with appropriate architectural modifications, diffusion models can be successfully applied to image clustering, and, in some cases, they outperform traditional deep clustering approaches.

These developments raise the question of how well diffusion models can be utilized to cluster tabular data. While a recently developed model, TabDDPM [12], successfully adapts diffusion models to handle tabular data, its application has been limited to generative tasks, such as imputation and synthetic data generation. To our knowledge, no existing work has systematically explored the use of diffusion models for clustering on tabular data. Based on this gap, this thesis addresses the following research questions:

Research Questions

- Is the TabDDPM architecture, originally designed for imputation and synthetic data generation, also effective for clustering tasks?
- How does a deep clustering framework for tabular data, utilizing a diffusion model architecture, compare to other deep clustering approaches in terms of clustering accuracy on benchmark tabular datasets?
- Does the proposed method exhibit particular strengths or weaknesses under specific dataset conditions?

To explore this open direction, this thesis proposes **CluTaD**, a diffusion-based deep clustering framework for tabular data. The method integrates ideas from three previously introduced models: ClusterDDPM [11], TabDDPM [12], and G-CEALS [5]. By combining diffusion-based representation learning, tabular-specific preprocessing, and a clustering-aware loss function, **CluTaD** aims to leverage the strengths of each component in a unified architecture.

Summary of Results

The proposed framework, **CluTaD**, was evaluated against 12 baseline algorithms across 16 benchmark tabular datasets, where it achieved the best overall performance in terms of clustering accuracy by securing the highest average rank among all tested approaches. Notably, on the Balance Scale dataset (Dataset 11), the model surpassed the previous state-of-the-art by 5.3%, increasing clustering accuracy from 69.3% to 74.6%. However, two primary limitations were also identified. First, the model struggled with high-dimensional data, for instance, on a dataset with 70 features, accuracy dropped to 54.7% compared to the 99.9% baseline, though performance improved significantly after applying dimensionality reduction. Second, the model collapsed into a single-cluster solution on heavily imbalanced datasets with only two clusters, resulting in an Adjusted Rand Index of zero. Further experiments on balanced versions of these datasets confirmed

that **CluTaD** can successfully find meaningful partitions in two-cluster settings when the class distribution is equalized. On these balanced versions, **CluTaD** outperformed classical baselines in 8 out of 10 cases. These findings suggest that while **CluTaD** is highly competitive for tabular clustering tasks, its current version is best suited for datasets with moderate dimensionality and balanced cluster structures.

Structure

The thesis begins with introducing the relevant theoretical background in Chapter 2. The definitions and relevant algorithms used in previous works and throughout the thesis are introduced. Among others, the G-CEALS algorithm is detailed. Demonstrated in its original paper as the best-performing clustering algorithm for tabular datasets, it serves as a foundation for the proposed methodology.

Chapter 3 provides a review of related work. First, the focus is on benchmark deep clustering methods. The fundamentals of the most well-known algorithms are discussed. The chapter also explores the applications of deep clustering techniques to tabular data, underscoring that it is an underexplored area. It is followed by discussing the past implementations of diffusion models on tabular data and for clustering tasks.

Chapter 4 introduces the proposed methodology, detailing the novel approach that integrates deep clustering and diffusion models for tabular data, with the framework supported by theoretical analysis and formal derivations.

Chapter 5 introduces the datasets used in the experiments.

Chapter 6 presents the experimental results, evaluates the performance of the proposed model, and offers a discussion of the findings.

Finally, Chapter 7 concludes the thesis with a summary of key contributions and suggests potential directions for future research.

2. Background

In this chapter, I introduce the key concepts and algorithms used throughout the thesis, following conventions from previous works.

2.1. Clustering

Clustering is one of the fundamental tasks in unsupervised learning, aiming to identify meaningful groups within data. Consider the following example for an illustration.

Imagine that you walk into a library where books are scattered randomly. No genres, no authors grouped together. Finding a novel would be extremely difficult. Clustering is how we bring order to such chaos. From recommendation systems (grouping users with similar preferences) to diagnosing diseases (identifying patient subgroups), clustering is present in many real-world systems.

Traditional clustering algorithms, such as k-means [13] and hierarchical clustering [14], have long been the librarians of data science, organizing data into the appropriate shelves. However, these methods are called shallow methods, because they rely on a critical assumption: that the features of the data and the used distance measure are already informative enough for grouping. For example, sorting books based on their authors, titles, and genres. But in practice, the raw data often lack such features, like a library where instead only the author's birth date, their previous book title, and a vague story description are known. Without meaningful descriptors, even the best librarian would struggle to sort the books. Likewise, clustering algorithms struggle when the available features fail to reflect the underlying structure of the data. To make effective clustering possible, one can first transform these raw features into representations that can capture the semantic structure.

2.1.1. Limitations of Shallow Clustering

The algorithms mentioned in the previous section are referred to as shallow methods, meaning that they do not involve feature transformations. Instead, these methods operate directly on the input features, assuming that the original feature space is sufficiently informative to reveal the underlying cluster structure. However, this assumption often fails in practice, particularly for high-dimensional, noisy, or non-linearly separable data. Consequently, shallow clustering algorithms frequently have poor performance on complex datasets.

A common strategy to address this limitation is to apply a separate feature transformation prior to clustering. Traditional techniques such as:

- Principal Component Analysis (PCA) [15]
- Kernel methods [16]
- Manifold learning (e.g., t-SNE [17], UMAP [18])

are often used to project the data into more meaningful representations. While these methods often improved clustering performance to some extent, they have two limitations. First, these transformations are limited in expressiveness, restricting their ability to capture complex, non-linear structures in the data. Second, the generated features are learned independently of the clustering objective, meaning they are not optimized to reveal the underlying cluster structure.

2.1.2. Deep Clustering

A more flexible solution is to incorporate deep neural networks (DNNs) for representation learning. DNNs can model complex, non-linear relationships in the data as well as extract representations specifically tailored for a given task, for example, grouping similar data points.

When combined with clustering objectives, they enable end-to-end learning of both feature representations and cluster assignments. This insight motivated the foundation of deep clustering (DC).

Formally, in DC, let

$$\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \quad \mathbf{x}^{(i)} \in \mathbb{R}^{d}$$

denote the original data features and

$$\mathbf{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{n}, \quad \mathbf{z}^{(i)} \in \mathbb{R}^{d'}$$

the latent features, with $d > d'$. We define an encoder

$$g_{\boldsymbol{\theta}_{\boldsymbol{E}}} : \mathbb{R}^{d} \to \mathbb{R}^{d'}$$

that maps data into the latent space, followed by a clustering function f. Depending on the model, this function is either

$$f : \mathbb{R}^{d'} \to \{1, \ldots, k\},$$

if it produces hard assignments, or

$$f : \mathbb{R}^{d'} \to \Delta^{k-1}$$

if it produces soft assignments in the form of probability distributions over k clusters, where

$$\Delta^{k-1} = \left\{ \mathbf{p} \in \mathbb{R}^{k} \,\middle|\, \mathbf{p}_j \geq 0,\ \sum_{j=1}^{k} \mathbf{p}_j = 1 \right\}$$

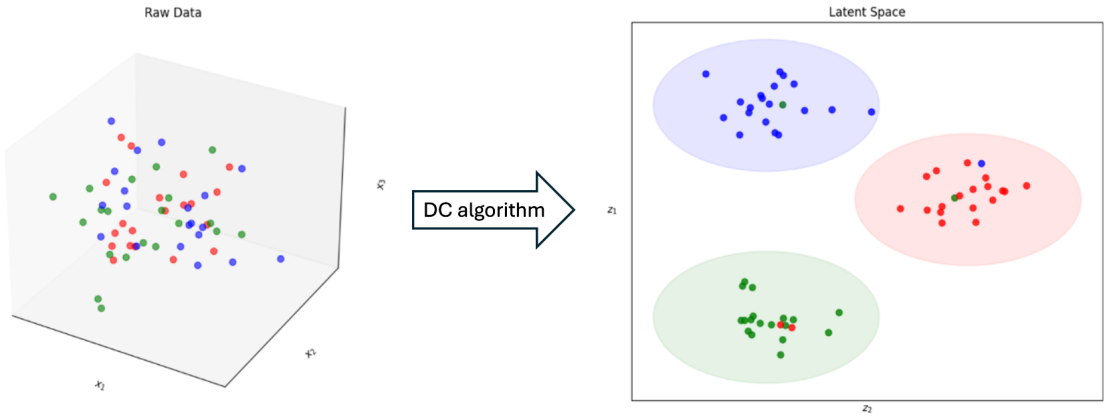

Figure 2.1.: Illustration of deep clustering: transforming high-dimensional raw data into a well-separated latent space

denotes the $(k-1)$-dimensional probability simplex. Thus, the cluster assignment of a data point $\mathbf{x}^{(i)}$ is given by $f(g_{\boldsymbol{\theta}_E}(\mathbf{x}^{(i)}))$.

One reason for the increase in clustering performance is that DC performs dimensionality reduction as part of the process, allowing clustering to be done on a lower-dimensional latent space rather than on data with potentially hundreds of raw features. This reduction is especially important for image data. As noted in the survey [19], DC methods are most commonly designed and evaluated on images, text, or audio, which typically have very high- dimensional feature spaces. In such cases, performing clustering on the reduced-dimensional latent representations is computationally efficient and might be beneficial for achieving optimal cluster structures, as illustrated in Figure 2.1.

2.2. Diffusion Models

One early approach for image generation involved training an AE, which can decode lower-dimensional representations back into the original feature space. In theory, this allows one to decode random vectors, which ideally result in realistic new data similar to the training set. However, in practice, this approach produces realistic outputs only if the random vector lies close to the encoded representations of real data [20]. Variational Autoencoders (VAEs) improved this by introducing a probabilistic framework that regularizes the latent space toward a continuous prior distribution, but they still often produce blurry outputs and struggle to accurately model complex data distributions [21].

To address the challenge of generating realistic data from arbitrary random inputs, diffusion models have recently emerged as a powerful alternative [22].

The concept of diffusion originates in thermodynamics, describing the movement of particles from regions of high concentration to low concentration through random motion. A classic example is a drop of ink dispersing in water until it reaches an equilibrium of uniform distribution (cf. Figure 2.2).

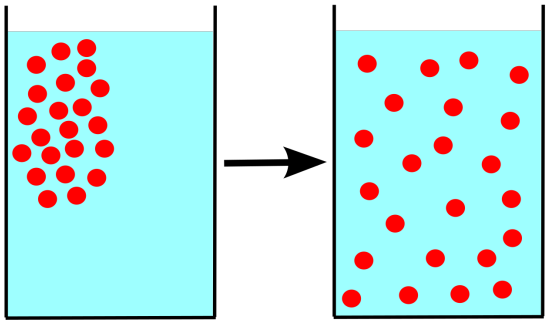

Figure 2.2.: Illustration of diffusion in thermodynamics: particles spreading from high to low concentration. Source: Wikipedia

In the context of generative modeling, we can draw a direct analogy to this physical process: we treat a data point, for example, an image, as the high concentration state and gradually add Gaussian noise until the structure is completely destroyed, reaching a state of equilibrium, in this case, pure noise (cf. Figure 2.3). This procedure is known as the forward diffusion process.

This idea, which first inspired the Diffusion Probabilistic Model [22], and later the Denoising Diffusion Probabilistic Model (DDPM) [9], reimagines data generation as the gradual restoration of structure, learning to reconstruct coherent samples from a simple noise distribution. This inverse procedure is known as the backward diffusion process, or denoising.

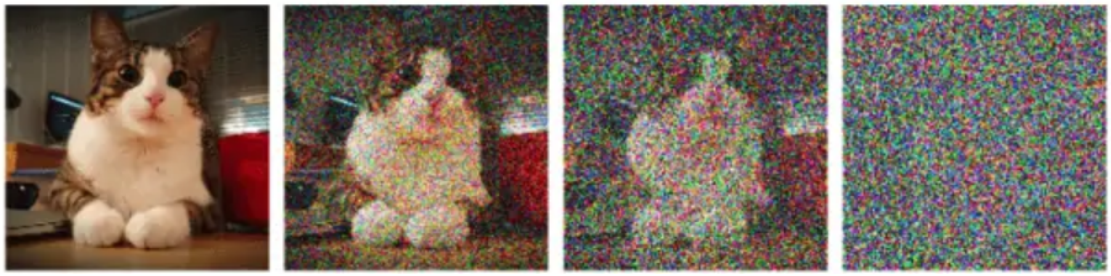

Figure 2.3.: Illustration of diffusion in generative modeling: the gradual addition of noise to transform structured data into pure noise. Source: NVIDIA Developer Blog

2.2.1. Diffusion Probabilistic Models

To formalize the intuition of viewing data generation as the reversal of a noise-addition process, we consider an unknown data distribution q, referred to as the target distribution, from which clean data instances are drawn: $\mathbf{x}_t^{(i)} \sim q(\mathbf{x}_t^{(i)})$, where $\mathbf{x}_t^{(i)}$ denotes the state of a data point i at timestep t with $t = 0$ representing the original signal and $t = T$

representing a state of total equilibrium, or pure noise. Our goal is to learn a model $p_\theta(\mathbf{x}_{\mathbf{0:T}}^{(\mathbf{i})})$ that can successfully reverse this transition, transforming samples from a simple prior distribution, typically the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, back into realistic samples from q.

Diffusion probabilistic models achieve this by defining a forward noising process that gradually adds noise to the data in T discrete time steps, transforming $\mathbf{x}_0$ into pure noise $\mathbf{x}_T^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. It can be defined as a Markov chain:

$$q(\mathbf{x}_t^{(i)} \mid \mathbf{x}_{t-1}^{(i)}) = \mathcal{N}\left(\mathbf{x}_t^{(i)}; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}^{(i)}, \beta_t \mathbf{I}\right),$$

where $\{\beta_t\}_{t=1}^T$ can be either learned or is a fixed variance schedule, and $\mathbf{I}$ denotes the identity matrix. The cumulative product of the values $\alpha_t = 1 - \beta_t$ is denoted as

$$\hat{\alpha}_t := \prod_{s=1}^{t} \alpha_s.$$

This forward process is beneficial because of two key properties:

1. As $t \to T$, the marginal distribution $q(\mathbf{x}_t^{(i)})$ approaches the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

2. The marginal $q(\mathbf{x}_t^{(i)} \mid \mathbf{x}_0^{(i)})$ can be expressed in a closed form:

   $$q(\mathbf{x}_t^{(i)} \mid \mathbf{x}_0^{(i)}) = \mathcal{N}\left(\mathbf{x}_t^{(i)}; \sqrt{\hat{\alpha}_t}\,\mathbf{x}_0^{(i)}, (1-\hat{\alpha}_t)\mathbf{I}\right),$$

   allowing direct sampling of $\mathbf{x}_t^{(i)}$ given $\mathbf{x}_0^{(i)}$ without having to iteratively apply the forward process.

Note that this forward diffusion process can be also expressed as

$$\mathbf{x}_t^{(i)} = \sqrt{\hat{\alpha}_t}\,\mathbf{x}_0^{(i)} + \sqrt{1-\hat{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.1}$$

The $p_\theta(\mathbf{x}_{0:T}^{(i)})$ backward diffusion process is defined the following way:

$$p(\mathbf{x}_T^{(i)}) = \mathcal{N}(\mathbf{x}_T^{(i)}; \mathbf{0}, \mathbf{I}),$$

$$p_\theta(\mathbf{x}_{0:T}^{(i)}) = p(\mathbf{x}_T^{(i)}) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}^{(i)} | \mathbf{x}_t^{(i)}),$$

where $p_\theta(\mathbf{x}_{t-1}^{(i)} \mid \mathbf{x}_t^{(i)})$ is one of step of the denoising process (see an example in Figure 2.4), and is generally also modeled as a Gaussian distribution parameterized by

$$p_\theta(\mathbf{x}_{t-1}^{(i)} \mid \mathbf{x}_t^{(i)}) = \mathcal{N}\left(\mathbf{x}_{t-1}^{(i)}; \mu_\theta(\mathbf{x}_t^{(i)}, t), \Sigma_\theta(\mathbf{x}_t^{(i)}, t)\right). \tag{2.2}$$

Here, μ_θ and Σ_θ represent the mean and covariance predicted by a deep neural network, which effectively helps to denoise. However, in practice, Σ_θ is often kept constant [9].

In this setup, the probability that the model assigns to the data is $p_\theta(\mathbf{x}_0^{(i)}) = \int p_\theta(\mathbf{x}_{0:T}^{(i)})\, d\mathbf{x}_{1:T}^{(i)}$. Hence, the training objective is to minimize the expected negative log-likelihood of the data distribution q under the model:

$$\mathbb{E}_{\mathbf{x}_0^{(i)} \sim q}\left[-\log p_\theta(\mathbf{x}_0^{(i)})\right].$$

Through variational inference and appropriate reformulations, this objective can be expressed as minimizing the following variational bound:

$$L = \mathbb{E}_q\left[L_T + \sum_{t=2}^{T} L_{t-1} + L_0\right], \tag{2.3}$$

where

$$\begin{aligned} L_T &= \mathrm{KL}\big(q(\mathbf{x}_T^{(i)} \mid \mathbf{x}_0^{(i)}) \| p(\mathbf{x}_T^{(i)})\big), \\ L_{t-1} &= \mathrm{KL}\big((q(\mathbf{x}_{t-1}^{(i)} \mid \mathbf{x}_t^{(i)}, \mathbf{x}_0^{(i)}) \| p_\theta(\mathbf{x}_{t-1}^{(i)} \mid \mathbf{x}_t^{(i)})\big), \\ L_0 &= -\log p_\theta(\mathbf{x}_0^{(i)} \mid \mathbf{x}_1^{(i)}). \end{aligned}$$

Here, $q(\mathbf{x}_t^{(i)} \mid \mathbf{x}_{t+1}^{(i)}, \mathbf{x}_0^{(i)})$ denotes the posterior distribution of the forward process, which is intractable but approximated via the learned reverse process. For more details, see the original paper [9].

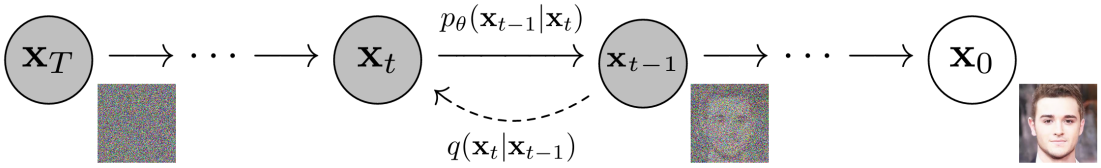

Figure 2.4.: Screenshot from the DDPM paper [9], illustrating the model

An important empirical finding was that breaking the denoising process into many small steps (i.e., large T) significantly improves training stability and sample quality [22]. Intuitively, each individual step becomes easier to learn, since it only needs to remove a small amount of noise. This incremental refinement allows the model to better approximate the data distribution over time.

With appropriately defined mean and covariance, the formulation of the backward process provides a tractable framework for training diffusion models, enabling gradient-based optimization.

2.2.2. Denoising Diffusion Probabilistic Models

While the general diffusion framework allows a broad class of backward processes, practical diffusion models require specific parametrizations of the mean and covariance [9]. These

constraints make training stable and computationally feasible. Denoising Diffusion Probabilistic Models (DDPMs) build on this idea by introducing a simplified yet effective formulation of the reverse process.

DDPM sets the covariance to a known time-dependent scalar multiple of the identity matrix:

$$\Sigma_\theta(\mathbf{x}_t^{(i)}, t) = \sigma_t^2 \mathbf{I},$$

where σ_t^2 is a predefined variance schedule that does not depend on $\mathbf{x}_t^{(i)}$.

The mean is parameterized as

$$\mu_\theta(\mathbf{x}_t^{(i)}, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t^{(i)} - \frac{\beta_t}{\sqrt{1-\hat{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t^{(i)}, t) \right),$$

where $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is a neural network that predicts the noise term $\boldsymbol{\epsilon}$ from the forward process in Equation (2.1).

This leads to a simplified training objective (2.3), where the model learns $\boldsymbol{\epsilon}$ by minimizing:

$$\mathbb{E}_{\mathbf{x}_0^{(i)}, \boldsymbol{\epsilon}} \left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t^{(i)}, t)||^2 \right].$$

The function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ was originally implemented as a U-Net architecture [23] due to its success in image-to-image translation tasks. More generally, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ can be any expressive deep neural network that preserves the dimensionality of the input, for example, an AE.

In summary, while the original formulation of diffusion probabilistic models [22] emphasized the importance of gradual denoising over many steps, the key insight of the DDPM paper [9] was that it is much more effective to train the model to predict the added noise $\boldsymbol{\epsilon}$, rather than to directly predict the original data $\mathbf{x}_0^{(i)}$. This meant a significant breakthrough, as it demonstrated remarkable image generation capabilities, surpassing many previous generative models.

2.3. Core Algorithms

In this section, we present the three core algorithms that form the foundation of **CluTaD**: ClusterDDPM, TabDDPM, and G-CEALS. All three methods were already introduced in the related work (Chapter 3), here, we describe them in more detail. Figure 2.5 presents the conceptual overlap between them. The formal pseudocode for each algorithm are also presented for clarity and reference.

2.3.1. ClusterDDPM: an EM clustering framework with Denoising Diffusion Probabilistic Models

ClusterDDPM [11][1] proposes a theoretically grounded framework for clustering using diffusion models. It combines the EM (Expectation-Maximization) algorithm with a

[1]The official Python implementation of ClusterDDPM is available at `https://github.com/Jarvisyan/ClusterDDPM-pytorch`.

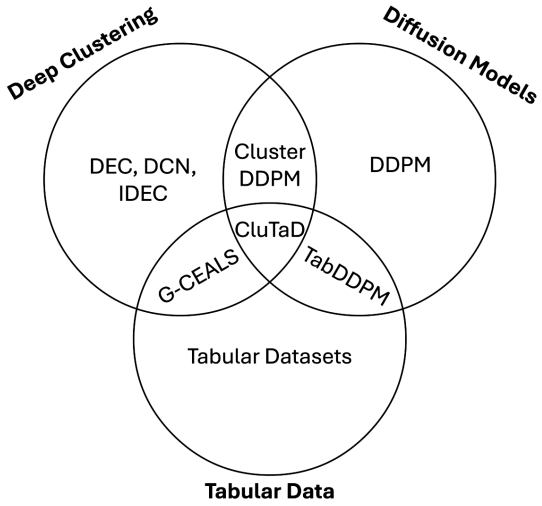

Figure 2.5.: Conceptual overlap of **CluTaD**'s components

conditional DDPM to iteratively refine both cluster assignments and latent representations through a combination of denoising and prior matching. In this section, let $\mathbf{x}_t^{(i)}$ denote again the i-th data point at time t. For simplicity, when referring to the initial data point without respect to time, we use the shorthand $\mathbf{x}^{(i)}$, i.e., $\mathbf{x}^{(i)} = \mathbf{x}_0^{(i)}$.

In diffusion models, there are many latent variables involved, which at first glance might seem to be potential embeddings suitable for clustering, however, they lie in the same space as the original data and primarily serve as intermediate transitions between data and noise. Hence, ClusterDDPM introduces a separate encoder network that maps an input data $\mathbf{x}^{(i)}$ into a latent representation $\mathbf{z}^{(i)}$. Afterwards, a Gaussian Mixture Model (GMM) [24] is trained on this latent space, resulting in soft cluster assignments for each sample. This GMM defines the cluster centroids $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^k$, the cluster covariance matrices $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_j\}_{j=1}^k$, and a prior over clusters, $\boldsymbol{\pi}$, with each component parameterized by a centroid and variance in latent space. This is the E-step of the EM loop.

During the M-step, the DDPM learns to reconstruct $\mathbf{x}^{(i)}$ from noisy observations $\mathbf{x}_t^{(i)}$ by predicting the noise added to it, conditioned on both $\mathbf{z}^{(i)}$ and the timestep t. This conditional reconstruction motivates the latent variables to be as informative as possible. The parameters of the encoder and the DDPM are updated based on the reconstruction loss:

$$\mathcal{L}_{\text{reconstruct}} = \|\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t^{(i)}, t, \mathbf{z}^{(i)}) - \boldsymbol{\epsilon}\|^2.$$

To encourage clustering-friendly representations, a prior matching loss is also introduced. This regularization term minimizes the KL divergence between the posterior of the encoder $q(\mathbf{z}^{(i)}, \boldsymbol{\pi} \mid \mathbf{x}^{(i)})$ and the GMM prior:

$$\mathcal{L}_{\text{cluster}} = \text{KL}\big(q(\mathbf{z}^{(i)}, \boldsymbol{\pi} \mid \mathbf{x}^{(i)}) \,\|\, p_{\text{GMM}}(\mathbf{z}^{(i)}, \boldsymbol{\pi})\big).$$

more details about the clustering loss can be found in the original paper [11]. This

alignment ensures that the learned latent space fits the cluster structure prescribed by the GMM, improving separability and interpretability.

The full training objective combines the standard diffusion loss with the prior matching term:

$$\mathcal{L}(\boldsymbol{\theta_E}, \boldsymbol{\theta_{DM}}, \boldsymbol{M}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \mathcal{L}_{\text{reconstruct}} + \lambda \cdot \mathcal{L}_{\text{cluster}},$$

where λ controls the strength of the regularization. The whole training process is further shown in Algorithm 1.

ClusterDDPM achieves state-of-the-art performance on several standard image clustering benchmarks (e.g., MNIST, Fashion-MNIST, CIFAR-10), outperforming both GAN-based and VAE-based approaches in terms of clustering accuracy and sample quality. Its modular architecture, consisting of a separate encoder and diffusion model, further enables flexible adaptation to different data modalities, including tabular data. However, to the best of our knowledge, such an extension has not yet been explored.

For **CluTaD**, ClusterDDPM demonstrates that diffusion-guided latent spaces, when regularized by prior matching, can support strong unsupervised clustering.

Algorithm 1: ClusterDDPM

Input: Dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$, $\mathbf{x}^{(i)} \in \mathbb{R}^d$, number of clusters k, and number of epochs *num_ epochs*

Output: Soft cluster assignments

Initialize encoder $g_{\boldsymbol{\phi}}$;
Initialize conditional DDPM $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$;
for *epoch* $= 1, \ldots,$ *num_ epochs* **do**
  `// E-step: Clustering in latent space`
  Encode samples: $\mathbf{Z} \leftarrow g_{\boldsymbol{\phi}}(\mathbf{X})$;
  Fit a GMM with k components on $\mathbf{Z}$ to obtain cluster priors $\boldsymbol{\pi}$, cluster centroids $\boldsymbol{M}$ and covariance matrices $\boldsymbol{\Sigma}$;
  `// M-step: Conditional DDPM and encoder training`
  for $i = 1, \ldots, n$ **do**
    $\mathbf{z}^{(i)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)})$;
    Get its cluster assignment c;
    Sample time step $t \sim \text{Uniform}(1, T)$ and noise $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;
    Apply Gaussian diffusion: $\mathbf{x}_t^{(i)} \leftarrow \sqrt{\hat{\alpha}_t}\mathbf{x}_0^{(i)} + \sqrt{1-\hat{\alpha}_t}\boldsymbol{\epsilon}_i$;
    Condition DDPM on $\mathbf{z}_i$: predict noise $\hat{\boldsymbol{\epsilon}}_i \leftarrow \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t^{(i)}, t, \mathbf{z}^{(i)})$;
  Compute total loss:
  $\mathcal{L} = \sum_{i=1}^{n} \left(\|\hat{\boldsymbol{\epsilon}}_i - \boldsymbol{\epsilon}_i\|^2 + \text{KL}\big(q(\mathbf{z}^{(i)}, \boldsymbol{\pi} \mid \mathbf{x}^{(i)}) \,\|\, p_{\text{GMM}}(\mathbf{z}^{(i)}, \boldsymbol{\pi})\big)\right)$;
  Update parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ via gradient descent;
return *Cluster assignments from GMM on* $g_{\boldsymbol{\phi}}(\mathbf{X})$

Trainable parameters for ClusterDDPM:

- Encoder network parameters: $\boldsymbol{\phi}$
- Denoising network parameters: $\boldsymbol{\theta}$
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$
- Cluster covariance matrices: $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_j\}_{j=1}^{k}$
- Cluster weight parameters: $\boldsymbol{\pi} = \{\pi_j\}_{j=1}^{k}$

2.3.2. TabDDPM: Modelling Tabular Data with Diffusion Models

Fang et al. [12] introduced TabDDPM[2], a denoising diffusion probabilistic model designed explicitly for mixed-type tabular data. The key idea of it is to employ different forward processes for different data types:

- **Continuous features** are transformed via a quantile transformation to approximate Gaussian marginals. Then, a standard diffusion process is applied to model the noise.
- **Categorical features** are encoded using one-hot vectors and modeled via a multinomial diffusion process [25], which gradually corrupts the categorical distribution by injecting uniform noise across classes.

In this section, let $\mathbf{x}_t^{(i)}$ denote the state of the i-th data point at time t. We represent its features as a concatenation of continuous and categorical components:

- $\mathbf{x}_{t,\text{cont}}^{(i)}$ represents the continuous features.
- $\mathbf{x}_{t,c}^{(i)}$ represents the probability vector for the c-th categorical variable ($c = 1, \ldots, C$), which has K_c categories, that is, $\mathbf{x}_{t,c}^{(i)} \in \mathbb{R}^{K_c}$.

The multinomial forward diffusion, which is applied to these categorical features, is defined by:

$$q(\mathbf{x}_{t,c}^{(i)} \mid \mathbf{x}_{t-1,c}^{(i)}) = \text{Cat}\left(\mathbf{x}_{t,c}^{(i)}; (1-\beta_t)\mathbf{x}_{t-1,c}^{(i)} + \beta_t/K_c\right), \tag{2.4}$$

$$q(\mathbf{x}_{T,c}^{(i)}) = \text{Cat}\left(\mathbf{x}_{T,c}^{(i)}; \mathbf{1}/K_c\right) \tag{2.5}$$

where $\beta_t \in [0, 1]$ is the noise schedule. This formulation ensures that as $t \to T$, $\mathbf{x}_{t,c}^{(i)}$ becomes nearly uniform, enabling a stable denoising process [25].

The architecture of the backward process step is based on a U-Net-style neural network [23], conditioned jointly on the timestep and a mask that indicates the feature type.

Training uses the standard DDPM objective for continuous features: mean squared error (MSE) between true and predicted Gaussian noise. For categorical features the KL divergence is used to compare the predicted distribution with the true one, as introduced in recent discrete diffusion variants [25].

Formally, the denoising network consists of the parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{cont}}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C)$, where the parameters $\boldsymbol{\epsilon}_{\boldsymbol{\theta}_{\text{cont}}}$ function is used on the continuous features $\mathbf{x}_{t,\text{cont}}^{(i)}$

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}_{\text{cont}}}(\mathbf{x}_{t,\text{cont}}^{(i)}, t) = \hat{\boldsymbol{\epsilon}}_i,$$

[2]The official Python implementation of TabDDPM is available at https://github.com/yandex-research/tab-ddpm.

and the $\boldsymbol{\epsilon}_{\boldsymbol{\theta}_c}$ function is used on the categorical feature c

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}_c}(\mathbf{x}_{t,c}^{(i)}, t) = \hat{\mathbf{x}}_{0,c}^{(i)}$$

for $c = 1, \ldots, C$. Putting these together we get

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\mathbf{x}_t^{(i)}, t\right) = \left(\hat{\boldsymbol{\epsilon}}_i,\ \hat{\mathbf{x}}_{0,1}^{(i)}, \ldots, \hat{\mathbf{x}}_{0,C}^{(i)}\right).$$

These steps, along with the other calculations, are presented in Algorithm 2.

Overall, TabDDPM provides a framework for modeling heterogeneous tabular data within the diffusion paradigm. Its architecture and training regime make it a compelling starting point for tasks such as synthetic data generation, missing data imputation, and, in the context of this thesis, representation learning and clustering. As a frequently cited and empirically validated method, TabDDPM establishes that diffusion models can indeed be successfully adapted to the tabular domain.

Algorithm 2: TabDDPM

Input: Tabular dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \quad \mathbf{x}^{(i)} \in \mathbb{R}^d$ with continuous and C categorical features, mini-batch size, and noise schedule β_t

Output: Trained denoising network $(\boldsymbol{\epsilon}_{\boldsymbol{\theta}_{\text{cont}}}, \boldsymbol{\epsilon}_{\boldsymbol{\theta}_1}, \ldots, \boldsymbol{\epsilon}_{\boldsymbol{\theta}_C})$ for synthetic sample generation

Preprocess continuous features using quantile transformation;
One-hot encode categorical features;
for *each mini-batch of indices* $\mathcal{B} \subset \{1, \ldots, n\}$ **do**
  for *all* $i \in \mathcal{B}$ **do**
    Split $\mathbf{x}_0^{(i)}$ into continuous part $\mathbf{x}_{0,\text{cont}}^{(i)}$ and categorical parts $\mathbf{x}_{0,c}^{(i)}$ for $c = 1, \ldots, C$;
    Apply Gaussian diffusion to $\mathbf{x}_{0,\text{cont}}^{(i)}$:

$$t \sim \text{Uniform}(1, T), \;\; \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\hat{\alpha}_t := \prod_{s=1}^{t} (1 - \beta_s)$$

$$\mathbf{x}_{t,\text{cont}}^{(i)} \leftarrow \sqrt{\hat{\alpha}_t}\, \mathbf{x}_{0,\text{cont}}^{(i)} + \sqrt{1 - \hat{\alpha}_t}\, \boldsymbol{\epsilon}_i;$$

    Apply multinomial diffusion to each $\mathbf{x}_{0,c}^{(i)}$:

$$\mathbf{x}_{t,c}^{(i)} \sim \text{Cat}(\hat{\alpha}_t \mathbf{x}_{0,c}^{(i)} + (1 - \hat{\alpha}_t)/K_c);$$

    Predict components:

$$\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_{\boldsymbol{\theta}_{\text{cont}}}(\mathbf{x}_{t,\text{cont}}^{(i)}, t)$$

$$\hat{\mathbf{x}}_{t,c}^{(i)} \leftarrow \boldsymbol{\epsilon}_{\boldsymbol{\theta}_c}(\mathbf{x}_{t,c}^{(i)}, t) \;\text{ for }\; c = 1, \ldots, C;$$

  Compute total loss:

$$\mathcal{L} = \sum_{i \in \mathcal{B}} \left(\underbrace{\|\hat{\boldsymbol{\epsilon}}_i - \boldsymbol{\epsilon}_i\|^2}_{\text{MSE}} + \underbrace{\frac{1}{C} \sum_{c=1}^{C} \text{KL}\big(\mathbf{x}_{0,c}^{(i)} \,\|\, \hat{\mathbf{x}}_{0,c}^{(i)}\big)}_{\text{Categorical KL divergence}} \right)$$

  Update $\boldsymbol{\theta}$ via gradient descent;
return *Trained model* $(\boldsymbol{\epsilon}_{\boldsymbol{\theta}_{cont}}, \boldsymbol{\epsilon}_{\boldsymbol{\theta}_1}, \ldots, \boldsymbol{\epsilon}_{\boldsymbol{\theta}_C})$

Trainable parameters for TabDDPM:

- Denoising network parameters: $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{cont}}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C)$

2.3.3. G-CEALS: Gaussian Cluster Assignment Learning

The G-CEALS [5][3] aims to directly tackle the challenges of DC on tabular data. The major structure of the algorithm is similar to that of other algorithms [2, 4]. A simple fully connected AE is used to generate embeddings, and the Cluster centroids are initialized using k-means applied to the embedding space produced by the AE before fine-tuning begins. Also, its training objective consists of a reconstruction loss, which is calculated from the inputs and outputs of the AE:

$$\mathcal{L}_{\text{recon}} = \frac{1}{n} \sum_{i} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2,$$

and a clustering loss, which is calculated from the soft cluster assignments $\mathbf{Q}$ and a target distribution $\mathbf{P}$:

$$\mathcal{L}_{\text{cluster}} = \lambda \cdot \sum_{i} \sum_{j} \mathbf{P}_{ij} \log \mathbf{Q}_{ij}.$$

However, there are some additional preprocessing steps, and the calculation of soft assignments and the target distribution differs from other approaches. Specifically, categorical features are one-hot encoded, and numerical features are standardized to handle heterogeneous data types effectively. Furthermore, unlike many DC algorithms that assume clusters in the latent space follow a Student's t-distribution, G-CEALS models assignments using Gaussian distributions. This choice is based on the fact that Gaussian distributions are less heavy-tailed, consequently, outliers have a smaller impact on the clustering process.

This approach also allows the model to learn the specific variances and sizes of the clusters, as instead of the typical t-distribution soft assignments common in image clustering, assignments are calculated using the Mahalanobis distance between data points and cluster centers. In detail, if given a latent data point $\mathbf{z}^{(i)}$ and a cluster j with centroid $\boldsymbol{\mu}_j$ and diagonal covariance matrix $\boldsymbol{\Sigma}_j$[4], then

$$d_{ij} = \sqrt{(\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)}. \tag{2.6}$$

By capturing distinct elliptical shapes, the algorithm can better represent the complex relationships between features typically found in tabular data.

They also address the issue of highly imbalanced tabular datasets [26, 27] by introducing an early stopping mechanism. This approach monitors the cluster weights during every training step, and if any weight falls below a predefined threshold, the training stops. This prevents the model from collapsing or merging smaller clusters into larger ones.

Another important novelty is that the authors hypothesized that guiding the latent cluster assignments $\mathbf{Q}$ via an independent target distribution $\mathbf{P}$, generated by a randomized

[3] The official Python implementation is available at `https://github.com/mdsamad001/G-CEALS---Deep-Clustering-for-Tabular-Data`.

[4] In the official implementation, a Softplus function ensures that all diagonal elements in the covariance matrices remain positive to guarantee positive definiteness.

MLP head, would prevent the predictions from collapsing into a wrong initial guess, as previously $\mathbf{P}$ was derived as a sharpened version of $\mathbf{Q}$. While it initially introduces a stochastic target, the empirical findings in [5] demonstrate that this guidance effectively encourages the model to discover more robust cluster structures. More specifically, the authors evaluated the model across 16 diverse tabular datasets from OpenML [28]. When compared against 11 other shallow and deep clustering algorithms, it performed best in terms of average ranking. While it secured the top accuracy on four datasets, its average rank of 2.9 was the lowest among all 12 methods.

Its performance is especially notable on the Balance Scale dataset (Dataset 11), which consists of weight and distance attributes. On this dataset, the best baseline method (DCN) reached 57.6% accuracy, while G-CEALS achieved 69.3%, a significant improvement of 11.7%.

The steps of the algorithm are summarized in the pseudocode shown in Algorithm 3. G-CEALS' novel approach allows it to better capture the heterogeneous distributions in tabular data. This contrasts with methods such as DEC or DCN, which rely on specific latent space assumptions that may not hold for tabular domains.

Furthermore, unlike many DC methods, it introduces cluster weight regularization to improve cluster stability and separation. In the paper, this is referred to as early stopping, designed to prevent clusters from merging during prolonged training (Figure 2.6).

Algorithm 3: G-CEALS

Input: Dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$, $\mathbf{x}^{(i)} \in \mathbb{R}^d$, number of clusters k, mini-batch size, and number of epochs

Output: Soft cluster assignments $\mathbf{Q} \in \mathbb{R}^{n \times k}$

Pretrain the autoencoder $(g_{\boldsymbol{\theta_E}}, g_{\boldsymbol{\theta_D}})$ by minimizing reconstruction loss on $\mathbf{X}$;
Initialize MLP classifier $f_{\boldsymbol{\theta_C}}$ for target assignment prediction;
Initialize cluster centroids $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$ via k-means on $g_{\boldsymbol{\theta_E}}(\mathbf{X})$;
Initialize $\boldsymbol{\Sigma}$ as $\mathbf{I}$;
for *epoch* $= 1, \ldots,$ *num_ epochs* **do**
  for *each mini-batch of indices* $\mathcal{B} \subset \{1, \ldots, n\}$ **do**
    for *all* $i \in \mathcal{B}$ **do**
      `// Forward pass`
      Encode latent representations: $\mathbf{z}^{(i)} \leftarrow g_{\boldsymbol{\theta_E}}(\mathbf{x}^{(i)})$;
      Reconstruct inputs: $\hat{\mathbf{x}}^{(i)} \leftarrow g_{\boldsymbol{\theta_D}}(\mathbf{z}^{(i)})$;
      `// Compute soft assignments`

$$d_{ij} \leftarrow \sqrt{(\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)};$$

$$\mathbf{Q}''_{ij} \leftarrow \exp\left(-\frac{1}{2} d_{ij}\right);$$

$$\mathbf{Q}'_{ij} \leftarrow \frac{\mathbf{Q}''_{ij}}{\sum_{l=1}^{k} \mathbf{Q}''_{il}};$$

$$\boldsymbol{\pi}_j \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{Q}'_{ij};$$

$$\mathbf{Q}_{ij} \leftarrow \boldsymbol{\pi}_j \cdot \mathbf{Q}'_{ij};$$

      `// Target distribution sharpening`

$$\mathbf{P}_{ij} \leftarrow \frac{\exp(f_{\boldsymbol{\theta_C}}(\mathbf{z}^{(i)})_j)}{\sum_l \exp(f_{\boldsymbol{\theta_C}}(\mathbf{z}^{(i)})_l)};$$

    `// Loss computation`
    Compute loss:

$$\mathcal{L} = \frac{1}{n} \sum_{i \in \mathcal{B}} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2 - \lambda \cdot \sum_{i \in \mathcal{B}} \sum_{j=1}^{k} \mathbf{P}_{ij} \log \mathbf{Q}_{ij};$$

    `// Update step`
    Update $\boldsymbol{\theta_E}$, $\boldsymbol{\theta_D}$, $\boldsymbol{\theta_C}$, $\boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$ via gradient descent;
    `// Early stopping check`
    if $\exists j : \boldsymbol{\pi}_j \leq \frac{1}{2k}$: stop training
return *Final soft cluster assignments* $\mathbf{Q}$

Trainable parameters for G-CEALS:

- Encoder weights and biases: $\boldsymbol{\theta_E}$
- Decoder weights and biases (used during pretraining only): $\boldsymbol{\theta_D}$
- MLP classifier for prediction sharpening: $\boldsymbol{\theta_C}$
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$
- Cluster covariance matrices: $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_j\}_{j=1}^{k}$

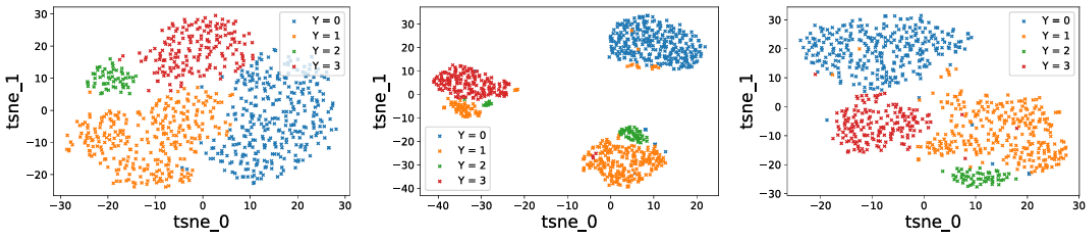

Figure 2.6.: The effect of early stopping. Source: [5]

3. Related Work

This chapter provides an overview of prior work relevant to the development of **CluTaD**. Section 3.1 introduces foundational deep clustering methods, namely DEC, DCN, and IDEC. These methods are commonly used as benchmarks in modern deep clustering literature.

Section 3.2 then explores clustering tabular data. Finally, Section 3.3 looks into how diffusion models have been extended towards representation learning and clustering tasks.

3.1. Foundational Deep Clustering Methods

Several foundational methods had a significant impact on the field of DC. Below, three of the most influential algorithms are discussed, which later on serve as benchmark methods [19].

3.1.1. DEC (Deep Embedding Clustering, 2016)

DEC [2] was one of the first successful approaches to combine deep learning with clustering. Instead of clustering directly on the raw input data, which is often high-dimensional and noisy, DEC first learns a lower-dimensional latent space using a denoising autoencoder (DAE). A DAE is a type of neural network consisting of an encoder and a decoder. Unlike a standard autoencoder that simply maps input vectors to a latent space, a DAE is fed noisy inputs, forcing the model to learn the underlying manifold structure to successfully repair the data. Consequently, the encoder part of the neural net maps each input to more compact latent representations where clustering becomes easier.

The method is trained in two phases. During the pretraining phase, only the DAE is trained by minimizing the reconstruction loss between the original inputs and the DAE outputs. This process initializes the encoder $g_{\boldsymbol{\theta_E}}$ with a meaningful data representation. Subsequently, the cluster centroids are initialized by running k-means on the latent representations produced by the encoder. Then, in the second phase, the decoder is discarded, as illustrated in Figure 3.1, and clustering is performed on the latent representations. Unlike methods such as k-means, which return a single hard assignment for each data point, DEC returns soft cluster assignments. This means that the model outputs a probability distribution where each value represents the likelihood of that data point belonging to a specific cluster. Let us store these soft cluster assignments in the matrix $\mathbf{Q} = [q_{ij}]$, where each q_{ij} is computed using an assignment mechanism based on the Student's t-distribution:

$$q_{ij} = \frac{(1 + \|\mathbf{z}^{(i)} - \boldsymbol{\mu}_j\|^2/\gamma)^{-\frac{\gamma+1}{2}}}{\sum_{j'}(1 + \|\mathbf{z}^{(i)} - \boldsymbol{\mu}_{j'}\|^2/\gamma)^{-\frac{\gamma+1}{2}}},$$

where $\mathbf{z}^{(i)}$ is the embedding of point $\mathbf{x}^{(i)}$, i.e., $\mathbf{z}^{(i)} = g_{\boldsymbol{\theta_E}}(\mathbf{x}^{(i)})$, $\boldsymbol{\mu}_j$ is the centroid of cluster j (with the set of all centroids denoted as $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^k$), and γ is the degrees of freedom (typically set to 1).

To guide the training process, a target distribution matrix $\mathbf{P} = [p_{ij}]$ is defined to refine and sharpen the current assignments. This target increases the influence of confident predictions and reduces noise from uncertain ones. It is computed as:

$$p_{ij} = \frac{q_{ij}^2 / \sum_{i'} q_{i'j}}{\sum_{j'} \left(q_{ij'}^2 / \sum_{i'} q_{i'j'} \right)}$$

The model is then trained by minimizing the Kullback-Leibler divergence between the soft cluster assignments $\mathbf{Q}$ and the target distributions $\mathbf{P}$. It measures how much $\mathbf{P}$ differs from $\mathbf{Q}$:

$$\mathrm{KL}(\mathbf{P}\|\mathbf{Q}) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

That is, the loss function is the following:

$$\mathcal{L}(\boldsymbol{\theta_E}, \boldsymbol{M}) = \mathrm{KL}(\mathbf{P}\|\mathbf{Q}).$$

Minimizing $\mathcal{L}$ simultaneously updates both the encoder's parameters and the cluster centroids, leading to cluster-friendly representations in the latent space and appropriate cluster centroids. After training, these centroids are fixed and used for inference.

Trainable parameters:

- Encoder weights and biases: $\boldsymbol{\theta_E}$
- Decoder weights and biases: $\boldsymbol{\theta_D}$ (only during pretraining)
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^k$

3.1.2. DCN (Deep Clustering Network, 2017)

DCN [3] follows a similar two-phase process as DEC, but takes a different direction when it comes to clustering optimization. Instead of discarding the decoder after pretraining, DCN keeps the entire autoencoder (AE) structure. The motivation is that having the reconstruction loss alongside clustering can help retain meaningful structure in the latent space, which improves clustering performance.

The architecture is based on the stacked autoencoder (SAE) [29] architecture, which learns lower-dimensional representations of data by stacking multiple encoders on top of each other. What also sets DCN apart is its loss function. First, it has a centroid-based clustering objective directly on the latent representations instead of calculating the

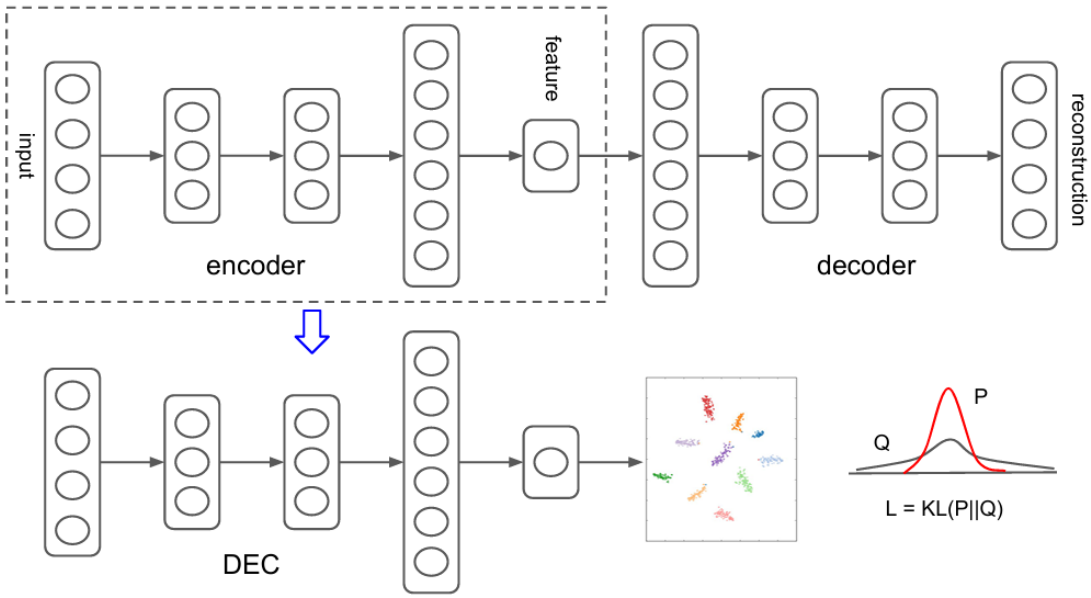

Figure 3.1.: Architecture of DEC. Source: [2]

Kullback-Leibler divergence. Second, DCN minimizes a joint loss function composed of two terms: the reconstruction loss, which ensures that the latent representation preserves enough information to approximately reconstruct the input, and the clustering loss, which encourages points in the latent space to group around cluster centroids. The full loss is:

$$\mathcal{L}(\boldsymbol{\theta_E}, \boldsymbol{\theta_D}, \boldsymbol{M}) = \sum_{i=1}^{n} \left(\|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\|_2^2 + \lambda \cdot \|\mathbf{z}^{(i)} - \boldsymbol{\mu}_{c(i)}\|^2 \right),$$

where $\mathbf{x}^{(i)}$ is the input, $\mathbf{z}^{(i)} = g_{\boldsymbol{\theta_E}}(\mathbf{x}^{(i)})$, $\hat{\mathbf{x}}^{(i)} = g_{\boldsymbol{\theta_D}}(\mathbf{z}^{(i)})$, $\boldsymbol{\mu}_{c(i)}$ is the centroid assigned to $\mathbf{z}^{(i)}$ (with the set of all centroids denoted as $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$), and λ is a balancing hyperparameter between the two terms.

Training is not via joint optimization, but by alternating between updating the AE weights (with cluster centroids fixed), followed by updating the cluster assignments and centroids (with weights fixed). Also note that this is a hard assignment approach as there is no matrix storing soft cluster assignments, and that inference is via assigning the final latent representations to the closest final cluster.

Trainable parameters:

- Encoder weights and biases: $\boldsymbol{\theta_E}$
- Decoder weights and biases: $\boldsymbol{\theta_D}$
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$

3.1.3. IDEC (Improved Deep Embedded Clustering)

IDEC [4] is an improved version of DEC that addresses the limitation of the original model in the same way as DCN. While DEC discards the decoder $g_{\boldsymbol{\theta_D}}$ after pretraining and only fine-tunes the encoder $g_{\boldsymbol{\theta_E}}$ with the clustering loss, here the decoder is kept throughout training.

Otherwise IDEC calculates the cluster assignments $\mathbf{Q}$ and the target distribution $\mathbf{P}$ the same way as DEC, keeps the KL divergence objective, just similarly to DCN it adds the $\|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\|^2$ reconstruction loss to the final loss function as another term:

$$\mathcal{L}(\boldsymbol{\theta_E}, \boldsymbol{\theta_D}, \boldsymbol{M}) = \sum_{i=1}^{n} \|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\|_2^2 + \lambda \cdot \mathrm{KL}(\mathbf{P}\|\mathbf{Q}),$$

where $\hat{\mathbf{x}}^{(i)} = g_{\boldsymbol{\theta_D}}(g_{\boldsymbol{\theta_E}}(\mathbf{x}^{(i)}))$ and λ is a balancing hyperparameter.

In practice, this approach typically achieves higher accuracy than other clustering algorithms. For instance, on the MNIST dataset [30] (which has 10 classes), IDEC achieved a clustering accuracy of 88%, outperforming the classical k-means (53%) and the two deep baselines discussed previously: DEC (87%) and DCN (83%). As a result, IDEC is often considered the most effective deep clustering model in the literature for image-based benchmarks. [7].

Trainable parameters:

- Encoder weights and biases: $\boldsymbol{\theta_E}$
- Decoder weights and biases: $\boldsymbol{\theta_D}$
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$

3.2. Deep Clustering for Tabular Data

DC methods have achieved outstanding performance on images and text, but their application to clustering problems on classical tabular datasets remains limited. Tabular data presents unique challenges due to its heterogeneity, combining numerical and categorical features, lack of spatial structure, and the frequent occurrence of smaller dataset sizes. These characteristics are reflected in Table 3.1, which is based on the datasets evaluated in [2, 4, 5, 31]. Heterogeneity here implies that, unlike image data, the features in tabular datasets often follow distinct and unrelated distributions [31]. These aforementioned properties often result in weaker or less well-defined cluster structure, making DC for tabular data substantially more complex [5, 7].

It is well-researched that, for tabular datasets, deep learning is yet to replace traditional machine learning [32, 33]. This conclusion, however, was mainly drawn from supervised tasks, leaving hope for exploration in unsupervised scenarios such as clustering.

Recently, a comprehensive survey by Ren et al. [19] was published reviewing over 100 deep clustering algorithms. However, none of these algorithms was specifically designed for tabular data. To the best of our knowledge, up to December 2025 only a limited

Table 3.1.: Comparison of tabular and image datasets

Characteristic	**Tabular Data**	**Image Data**
Heterogeneity	Heterogeneous variables	Homogeneous variables
Sample Size	typically $<$ 10,000	typically $>$ 10,000
Dimensionality	typically $<$ 200	typically $>$ 250

number of studies have proposed deep clustering algorithms specifically optimized for tabular data. The most relevant among them introduced the G-CEALS algorithm [5], which is described in Section 2.3.3 as it is a building block of this work. Two other works are presented below.

3.2.1. TableDC: Deep Clustering for Tabular Data

TableDC [34] is a clustering framework that groups data at multiple levels: it can cluster not only individual rows, but columns and entire tables as well. It employs the Mahalanobis distance (see Equation (2.6)) to account for feature variances and correlations. TableDC adopts a loss function similar to IDEC and states that k-means as the cluster initialization on the latent vectors is not optimal for tabular data.

However, it was pointed out in the G-CEALS paper [5] that

"the deep learning literature for tabular data is entirely focused on supervised classification tasks [...] with the exception of table deep clustering (TableDC). However, TableDC is benchmarked on specialized web content and text data from various websites and uses large language models (LLMs) to learn mostly textual content structured in tables."

This observation highlights that while TableDC is a step towards deep clustering for structured data, its focus remains primarily on semi-structured, text-heavy tables, rather than general-purpose tabular data, which is the scope of this work.

3.2.2. IDC: Interpretable Deep Clustering for Tabular Data

The paper by Svirsky and Lindenbaum introduces IDC (Interpretable Deep Clustering) [35], which is also a deep clustering framework designed for tabular data, but with a strong focus on the interpretability of the predictions. To address this, IDC has a two-stage process.

In the first stage, an AE learns to reconstruct each sample using only a sparse subset of its features. This forces the model to identify the most informative ones for each instance individually via reducing dimensionality. In the second stage, a clustering head is trained jointly with a global gating matrix, which highlights feature relevance on a cluster level.

By combining reconstruction, sparsity, and clustering losses, IDC achieves both high-quality partitions and interpretability.

However, a key characteristic of the considered tabular datasets is that they often contain far more features than samples, making feature selection the most essential step of the model. While this makes IDC particularly suitable for domains where the curse of

dimensionality is severe, it is not the focus of this work. Hence, its mechanism will not be used.

3.3. Relevant Works with Diffusion Models

Diffusion models have gained popularity in generative modeling due to their ability to model complex data distributions via iterative denoising of random noise. In this section, prior works are reviewed that use diffusion models in contexts relevant to this thesis, namely clustering and tabular data, highlighting their applications and limitations.

3.3.1. Diffusion Models and Tabular Data

Similarly to deep clustering algorithms, most diffusion model research to date has been centered around fully continuous domains such as images and audio. In the case of diffusion models, this preference is largely driven by their reliance on Gaussian noise assumptions and continuous formulations that underlie standard diffusion processes. In contrast, extending diffusion models to tabular data introduces several unique challenges stemming from the reasons already mentioned in Section 3.2: tabular datasets often include both continuous and categorical variables, complex marginal distributions, and a tendency towards small sample sizes.

Recent works have begun addressing these challenges by adapting the diffusion framework to handle structured tabular data. For instance, Zheng and Charoenphakdee [36] proposed a diffusion model for imputation and generation tasks, while Gao et al. [37] also introduced a related approach. In the context of data generation, diffusion models have been already applied to specific domains, such as FinDiff [38] for financial data. The most prominent contribution, however, is TabDDPM [12], which is explained in detail in Section 2.3.2 as it is the second building block of our model.

3.3.2. Diffusion Models in Unsupervised Learning

Beyond data generation, recent research has increasingly explored the potential of diffusion models for unsupervised representation learning and clustering of image datasets.

For instance, Diffusion Autoencoders [39] propose an autoencoding framework in which an input $\mathbf{x}_0^{(i)}$ is mapped into a semantic latent vector $\mathbf{z}^{(i)}$ using an encoder. This latent vector is then used to condition both the forward diffusion process ($\mathbf{x}_0^{(i)} \to \mathbf{x}_T^{(i)}$) and the reverse denoising process ($\mathbf{x}_T^{(i)} \to \hat{\mathbf{x}}_0^{(i)}$). By forcing the model to reconstruct $\mathbf{x}_0^{(i)}$ from noise conditioned on $\mathbf{z}^{(i)}$, the latent representation is encouraged to capture high-level semantics, while the noise trajectory accounts for low-level details. Empirical results demonstrate that the learned $\mathbf{z}^{(i)}$ vectors are meaningful and well-suited for downstream clustering tasks.

In another work, Kim et al. [40] also showed that semantic information in images can be captured by training encoders jointly with diffusion processes.

Several other works even integrate clustering into the diffusion process itself. Structured Generations [41] incorporates hierarchical clustering into the dynamics of diffusion models, allowing data generation for a given cluster. Similarly, DiFiC [10] introduces a conditional DDPM framework that iteratively refines pseudo-labels and optimizes the generative process to align with latent class structure resulting in a competitive clustering algorithm.

Together, these works establish a foundation for employing diffusion models as a tool for clustering. Building on these insights, the most relevant contribution for this thesis is ClusterDDPM [11], a method that introduces a clustering objective into the diffusion training process. This method is further detailed in Section 2.3.1 as the final building block of **CluTaD**.

4. Methodology

This chapter presents the proposed **CluTaD** (Clustering Tabular Data with Diffusion) framework. Our goal was to design a clustering algorithm specifically tailored for tabular data, while leveraging recent advances both in diffusion models and deep clustering. The proposed framework is built upon three existing models: ClusterDDPM, TabDDPM, and G-CEALS, each being at the intersection of two out of the three domains: deep clustering, diffusion models, and tabular data.

Before implementing **CluTaD**, as an intermediate verification step, a simple version of the model was implemented, called **CluTaD-0**. In this simple version, two out of the three core algorithms, ClusterDDPM and TabDDPM, were put together in order to confirm that adapting ClusterDDPM's framework to tabular data is feasible and results in meaningful clustering behaviour (see Appendix A.2).

4.1. Overview of the Core Algorithms

To motivate the design choices behind **CluTaD**, first, we summarize the three models on which it is built. These algorithms were discussed in more detail in Section 2.3. Each of them contributes complementary strengths: ClusterDDPM integrates diffusion models with clustering, TabDDPM adapts diffusion models to heterogeneous feature spaces, and G-CEALS introduces a clustering loss specifically for tabular data. Together, they cover the pairwise intersections among the three central concepts of this work.

- **ClusterDDPM** combines diffusion models with clustering through an EM-style training scheme, achieving strong performance on image data. However, its diffusion process and loss formulation assume fully continuous inputs, making it unsuitable for tabular datasets that include categorical features.
- **TabDDPM** extends diffusion models to the tabular domain by incorporating heterogeneous forward and reverse processes: a Gaussian diffusion model is used for numerical variables and multinomial diffusion models are used for categorical variables. Although it was originally developed for data generation, it can also learn latent representations that capture meaningful data structure, which can be used for clustering. This was validated by additional experiments (see Appendix A.1).
- **G-CEALS** is a state-of-the-art deep clustering algorithm designed specifically for tabular data. It introduces a Gaussian-based clustering loss and a training procedure that handles imbalanced tabular structures. However, it does not use diffusion models and thus lacks the strong latent representations that diffusion-based methods can provide.

Bringing these ideas together, our central hypothesis is that combining the structural advantages of ClusterDDPM, the specialized handling of heterogeneous data in TabDDPM, and the clustering-specific optimization of G-CEALS results in a framework that demonstrates competitive performance in clustering tabular data.

4.2. CluTaD

Our deep clustering algorithm aims to learn latent representations of tabular data specifically designed for an internal GMM to return high-quality clustering results. While a similar approach, ClusterDDPM, is highly competitive on image data, its architecture and underlying assumptions do not translate effectively to the tabular domain. To bridge this gap, **CluTaD** extends ClusterDDPM by adapting its strengths to the unique constraints of tabular features.

4.2.1. Model Architecture at Inference

At inference time, the model consists of two main components: a fully connected encoder that maps tabular inputs to a latent space, and a GMM that assigns soft cluster assignments based on these latent representations. Formally, given a dataset

$$\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \quad \mathbf{x}^{(i)} \in \mathbb{R}^d$$

, the model consists of:

- **Encoder network** $g_{\boldsymbol{\phi}}$, parameterized by $\boldsymbol{\phi}$, which is a mapping

  $$g_{\boldsymbol{\phi}} : \mathbb{R}^d \to \mathbb{R}^{d'}$$

  from the d-dimensional input space to an d'-dimensional latent space, where $d > d'$, and let $\mathbf{z}^{(i)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)})$.

- **Gaussian Mixture Model**, parameterized by
  - **Cluster centroids:** $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$,
  - **Cluster covariance matrices:** $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_j\}_{j=1}^{k}$,
  - **Mixing coefficients:** $\boldsymbol{\pi} = \{\pi_j\}_{j=1}^{k}$,

  which define the posterior cluster assignment probabilities.

For a given input $\mathbf{x}^{(i)}$, the inference process follows a deterministic path. The encoder first computes the latent embedding $\mathbf{z}^{(i)}$. Subsequently, the GMM computes the soft assignment $\mathbf{Q}_{ij}$ of $\mathbf{z}^{(i)}$ belonging to cluster j the following way:

$$\mathbf{Q}'_{ij} = \exp\left(-\frac{1}{2}\sqrt{(\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)}\right)$$

$$\mathbf{Q}_{ij} = \frac{\mathbf{Q}'_{ij}}{\sum_{l=1}^{k} \mathbf{Q}'_{il}}$$

The final hard cluster assignment c_i is then determined by selecting the component with the highest responsibility: $c_i = \arg\max_j Q_{ij}$.

While the inference-time architecture is structurally simple, the central challenge lies in learning encoder parameters that return latent representations suitable for clustering tabular data. To address this, the training architecture extends this simple encoder-and-GMM pipeline with additional components that guide the latent space towards a cluster-friendly structure.

4.2.2. Reconstruction Loss

In order to encourage the latent representations to retain information about the original data during training, we add a noising step and a corresponding denoising network to the architecture. This was motivated by the architecture of ClusterDDPM, where the model learns to reconstruct clean inputs from their noised versions while being conditioned on their latent representations. Intuitively, if during training the model receives a noised sample together with its latent embedding and is tasked to recover the original input, then the embedding is incentivized to encode as much relevant information as possible and to simultaneously recover the original input from noised variants, thus exposing it to a wider distribution of inputs.

In the following, let $\mathbf{x}_t^{(i)}$ denote the i-th data point at timestep t, with $\mathbf{x}_0^{(i)} = \mathbf{x}^{(i)}$.

Given a data point $\mathbf{x}_0^{(i)}$, the forward noising process produces $\mathbf{x}_t^{(i)}$ by adding random Gaussian noise to the numerical features. However, since ClusterDDPM was developed for image data, it assumes all inputs are continuous. To adapt the method for tabular data with mixed feature types, we replace the original diffusion model with the TabDDPM architecture, which introduces heterogeneous forward and reverse processes. Specifically, the forward process applies Gaussian diffusion to numerical features and discrete multinomial diffusion to categorical features.

Hence, during training, the model architecture adds:

- **Denoising network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$**, parametrized by $\boldsymbol{\theta}$, reconstructs data from noised inputs $\mathbf{x}_t^{(i)}$ conditioned on $\mathbf{z}^{(i)}$.

This denoising network receives the noised sample $\mathbf{x}_t^{(i)}$ concatenated with its latent representation $\mathbf{z}^{(i)}$ and predicts both the continuous noise and the categorical reconstruction:

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}\Big([\mathbf{x}_t^{(i)}, \mathbf{z}^{(i)}], t\Big) = \Big(\hat{\boldsymbol{\epsilon}}_i,\ \hat{\mathbf{x}}_{0,1}^{(i)}, \ldots, \hat{\mathbf{x}}_{0,C}^{(i)}\Big),$$

where $\hat{\boldsymbol{\epsilon}}_i$ denotes the predicted noise for the continuous features, and $\hat{\mathbf{x}}_{0,c}^{(i)}$ denotes the reconstructed categorical feature c.

As a result, the reconstruction-based component of the loss function is given by

$$\mathcal{L}_{\text{recon}}(\phi, \theta) = \sum_i \left(\|\hat{\boldsymbol{\epsilon}}_i - \boldsymbol{\epsilon}_i\|^2 + \frac{1}{C} \sum_{c=1}^{C} \text{KL}\Big(\mathbf{x}_{0,c}^{(i)} \,\|\, \hat{\mathbf{x}}_{0,c}^{(i)}\Big) \right). \tag{4.1}$$

A detailed derivation of this loss is provided in Section 2.3.2, since it directly follows the TabDDPM reconstruction objective.

To be able to use this architecture, certain preprocessing steps on both the numerical and categorical features needed to be taken, which are discussed in detail in Section 5.2.

4.2.3. Clustering Loss

To encourage the representations to be also cluster-friendly, most deep clustering algorithms use a sharpening technique, which is meant to make the clustering probabilities more confident. During our training procedure, we use the same method that was used in the G-CEALS paper, that is, adding an MLP layer ($h_{\boldsymbol{\psi}}$) that maps the latent representations into a target distribution. While this layer initially provides an arbitrary target, their empirical findings demonstrated that such a guidance effectively encourages the model to discover more robust cluster structures, and prevents a potential collapse into a wrong initial guess.

- **MLP head** $h_{\boldsymbol{\psi}}$, parametrized by $\boldsymbol{\psi}$, which is a mapping

$$h_{\boldsymbol{\psi}} : \mathbb{R}^{d'} \to \Delta^{k-1}$$

  from the d'-dimensional latent space into cluster distributions over k clusters.

In parallel, the actual soft assignments are computed from the GMM parameters using the Mahalanobis distance. The cross-entropy between these two distributions is then incorporated into the final loss, guiding the encoder to produce cluster-friendly features while encouraging the MLP classifier to generate more confident assignments.

Formally, if given a set of samples $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)})$, and k clusters, first soft assignments are computed from the latent variables:

$$\mathbf{z}^{(i)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)})$$

$$\mathbf{Q}'_{ij} = \exp\left(-\frac{1}{2}\sqrt{(\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)}\right)$$

$$\mathbf{Q} = (\mathbf{Q}_{ij})_{i=1,\ldots,n;\ j=1,\ldots,k}, \qquad \mathbf{Q}_{ij} = \frac{\mathbf{Q}'_{ij}}{\sum_{l=1}^{k} \mathbf{Q}'_{il}}$$

The target distributions are also computed:

$$\mathbf{P} = (\mathbf{P}_{ij})_{i=1,\ldots,n;\ j=1,\ldots,k}, \qquad \mathbf{P}_{ij} = \frac{\exp(h_{\boldsymbol{\psi}}(\mathbf{z}^{(i)})_j)}{\sum_l \exp(h_{\boldsymbol{\psi}}(\mathbf{z}^{(i)})_l)}$$

From these values, the clustering loss is derived:

$$\mathcal{L}_{\text{cluster}}(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\psi}) = -\frac{1}{n} \sum_i \sum_j \mathbf{P}_{ij} \log \mathbf{Q}_{ij}$$

4.2.4. Training Process

The overall training objective is composed of the reconstruction loss and the clustering-oriented loss:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda \cdot \mathcal{L}_{\text{cluster}},$$

where λ is a balancing hyperparameter controlling the interaction between the two terms.

The training procedure of **CluTaD** closely follows the EM-style structure used in ClusterDDPM, with some modifications to ensure compatibility with tabular data. In particular, the M-step is adapted to incorporate the heterogeneous diffusion process of TabDDPM.

The training cycle is illustrated in Figures 4.1 and 4.2, which visualize the E-step and the M-step, respectively. The two steps are defined as follows:

- **E-step:**
  - Encode all samples into latent representations using g_{ϕ}.
  - Fit a GMM with k components on the latent embeddings.

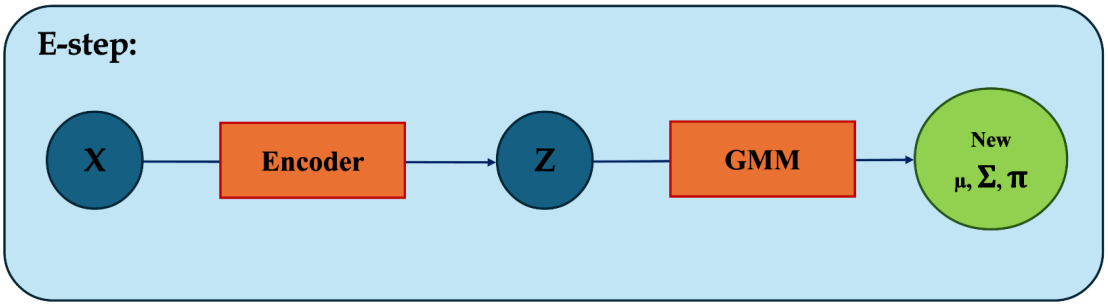

Figure 4.1.: E-step of **CluTaD**

- **M-step:**
  - *Reconstruction loss:*
    - Apply the heterogeneous noising process.
    - Use the denoising model $\boldsymbol{\epsilon_{\theta}}$, conditioned on the latent representations, to reconstruct the original inputs.
    - Compute the reconstruction loss between predicted and original values.
  - *Clustering loss:* Compute the clustering-oriented objective using the soft assignments and the h_{ψ} MLP classifier predictions.
  - Update the model parameters $\boldsymbol{\theta}$, $\boldsymbol{\psi}$, and $\boldsymbol{\phi}$ based on the total loss.

Note that although the MLP classifier h_{ψ} is used during training to shape the latent space, it is not used at inference time. Final cluster assignments are computed solely from the GMM fitted on the latent representations. The complete pseudocode of the **CluTaD** training procedure is provided in Algorithm 4.

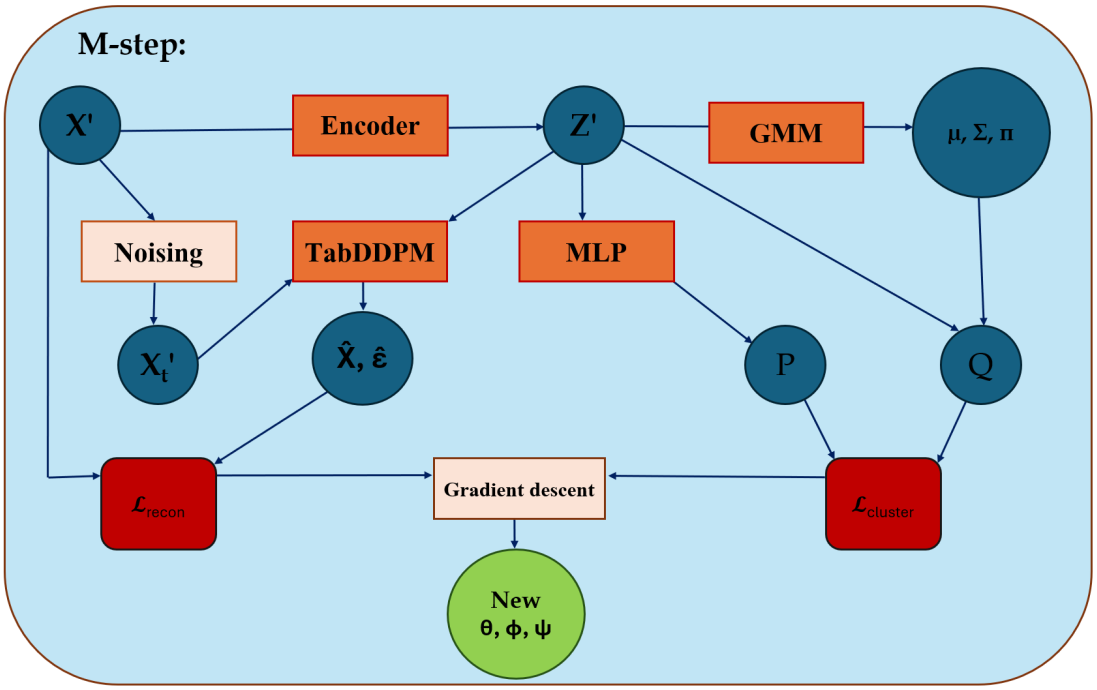

Figure 4.2.: M-step of **CluTaD**

4.2.5. Early Stopping

One limitation of ClusterDDPM is its lack of a mechanism for handling cluster imbalance or preventing cluster collapse. Unlike its benchmark datasets, where samples are typically balanced across categories [30], tabular datasets often have a natural imbalance. In many real-world tabular problems, such as medical diagnosis, fraud detection, or demographic surveys, certain classes occur far less frequently than others [26, 27]. To mitigate cluster collapse, G-CEALS [5] introduced an early stopping criterion based on the prior weights of the clusters. Training is terminated whenever any cluster's prior weight falls below $\frac{1}{2k}$, where k is the number of clusters. We also add this criterion in **CluTaD**.

However, one can argue that the fixed $1/2k$ early stopping criterion is disproportionately restrictive for small values of k. Specifically, for $k = 2$, a threshold of 25% leaves an insufficient margin to distinguish between a true numerical collapse and the inherent class imbalance typical of tabular data. In a binary setting, a cluster representing 20% of the data is often a highly informative, well-defined group, yet the $1/2k$ rule would trigger a termination. For example, in our benchmark dataset collection (see Chapter 6) there are 10 datasets with only $k = 2$ clusters, and in 7 of them the majority class accounts for at least 75% of the samples. For $k = 3$ and $k = 4$, the thresholds drop to 16.6% and 12.5% respectively. At these levels, our rule prevents a cluster collapse without too often stopping falsely due to the moderate imbalances in multi-class tabular data. Consequently, in **CluTaD**, this early stopping condition is only considered whenever $k > 2$.

Algorithm 4: CluTaD

Input: Dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \quad \mathbf{x}^{(i)} \in \mathbb{R}^d$, number of clusters k, mini-batch size, and number of epochs *num_ epochs*

Output: Soft cluster assignments

Initialize encoder $g_{\boldsymbol{\phi}}$, MLP classifier $h_{\boldsymbol{\psi}}$, and TabDDPM denoiser $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$;

for *epoch* $= 1, \ldots,$ *num_ epochs* **do**

`// E-step: Clustering in latent space`

Encode samples: $\mathbf{Z} \leftarrow g_{\boldsymbol{\phi}}(\mathbf{X})$;

Fit a GMM with k components on $\mathbf{Z}$;

`// M-step: Conditional TabDDPM and representation learning`

for *each mini-batch of indices* $\mathcal{B} \subset \{1, \ldots, n\}$ **do**

for *all* $i \in \mathcal{B}$ **do**

Compute soft assignments:

$$\mathbf{z}^{(i)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)});$$

$$\mathbf{Q}'_{ij} = \exp\left(-\frac{1}{2}\sqrt{(\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{z}^{(i)} - \boldsymbol{\mu}_j)}\right);$$

$$\mathbf{Q}_{ij} = \frac{\mathbf{Q}'_{ij}}{\sum_{l=1}^{k} \mathbf{Q}'_{il}};$$

$$\mathbf{Q} = (\mathbf{Q}_{ij})_{i=1,\ldots,n;\ j=1,\ldots,k};$$

Compute target distribution:

$$\mathbf{P}_{ij} = \frac{\exp(h_{\boldsymbol{\psi}}(\mathbf{z}^{(i)})_j)}{\sum_l \exp(h_{\boldsymbol{\psi}}(\mathbf{z}^{(i)})_l)};$$

$$\mathbf{P} = (\mathbf{P}_{ij})_{i=1,\ldots,n;\ j=1,\ldots,k};$$

Sample time step t and noise $\boldsymbol{\epsilon}$;

Apply heterogeneous diffusion to obtain $\mathbf{x}_t^{(i)}$;

Get TabDDPM predictions:

$$\left(\hat{\boldsymbol{\epsilon}}_i,\ \hat{\mathbf{x}}_{0,1}^{(i)}, \ldots, \hat{\mathbf{x}}_{0,C}^{(i)}\right) \leftarrow \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left([\mathbf{x}_t^{(i)}, \mathbf{z}^{(i)}], t\right);$$

Compute reconstruction loss:

$$\mathcal{L}_{\text{recon}} = \sum_{i \in \mathcal{B}} \left(\underbrace{\|\hat{\boldsymbol{\epsilon}}_i - \boldsymbol{\epsilon}_i\|^2}_{\text{MSE}} + \underbrace{\frac{1}{C} \sum_{c=1}^{C} \text{KL}\left(\mathbf{x}_{0,c}^{(i)} \,\|\, \hat{\mathbf{x}}_{0,c}^{(i)}\right)}_{\text{Categorical KL divergence}} \right);$$

Compute clustering loss: $\mathcal{L}_{\text{cluster}} = -\frac{1}{n} \sum_{i \in \mathcal{B}} \sum_{j=1}^{k} \mathbf{P}_{ij} \log \mathbf{Q}_{ij}$;

Compute batch loss: $\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda \cdot \mathcal{L}_{\text{cluster}}$;

Update parameters $\boldsymbol{\theta}$, $\boldsymbol{\psi}$, and $\boldsymbol{\phi}$ via gradient descent;

`// Early stopping criterion`

if $\exists j : \pi_j \leq \frac{1}{2k}$ and $k > 2$ **then** stop training;

return *Final cluster assignments from GMM on* $g_{\boldsymbol{\phi}}(\mathbf{X})$

4.3. Key Differences from the Core Algorithms

As mentioned before, **CluTaD** is built directly upon ClusterDDPM, TabDDPM, and G-CEALS. Conceptually, the overall structure remains the same as in ClusterDDPM, and in particular, the E-step is unchanged (see Figure 4.1). The encoder produces latent representations, and a GMM is fitted on these latent features in exactly the same way as in the original formulation.

The differences are entirely in the M-step, where two key modifications were introduced to make the model suitable for heterogeneous tabular data:

1. Tabular diffusion instead of continuous diffusion.

ClusterDDPM assumes continuous inputs and therefore uses a standard DDPM as the denoising model. This assumption is reasonable for image data but not for tabular datasets, which contain a mixture of numerical and categorical variables. To address this, **CluTaD** replaces the original DDPM denoiser with the TabDDPM architecture. TabDDPM introduces heterogeneous forward and reverse processes, Gaussian diffusion for numerical features and multinomial diffusion for categorical ones, allowing the reconstruction objective to meaningfully handle tabular inputs during training.

2. Tabular data-friendly clustering loss via G-CEALS.

ClusterDDPM inherits its clustering loss from DEC-style objectives, which rely on a sharpening mechanism based on the Student's t-distribution. Although effective for image domains, this has been shown to perform poorly on tabular data [5]. Following the findings of G-CEALS, **CluTaD** replaces the original sharpening-based objective with one that better captures the heterogeneous clusters typical of tabular data. Instead of constructing the target distribution $\mathbf{P}$ by sharpening the soft assignments $\mathbf{Q}$, an auxiliary MLP classifier h_{ψ} maps latent representations to soft cluster assignments, while the GMM provides a second, independent set of assignments based on the Mahalanobis distance. The cross-entropy between these two distributions serves as the new clustering loss, encouraging the encoder to learn cluster-friendly latent representations.

In summary, **CluTaD** keeps the EM structure and latent-space clustering of ClusterDDPM, but modifies the M-step in two crucial ways: (i) a tabular-aware diffusion process, namely TabDDPM, replaces continuous DDPM, and (ii) the clustering loss is replaced with the G-CEALS objective to better align with the statistical characteristics of tabular data.

5. Experimental Setup

This chapter provides an overview of the datasets used in the experiments, and explains the preprocessing steps taken to make them fit the proposed framework. Furthermore, it presents the evaluation metrics and the hyperparameter configurations used for model training and evaluation.

5.1. Datasets

To evaluate our proposed method, 16 tabular datasets from the OpenML repository [28] were used. OpenML is a collaborative platform that provides standardized, publicly available datasets along with detailed meta-information, making it widely used for benchmarking machine learning algorithms. The chosen datasets are publicly available, contain no duplicates, and have no missing values, which is crucial for clustering tasks where data imputation could bias the results. The choice of datasets follows the G-CEALS paper, allowing easy comparison with the most relevant state-of-the-art clustering algorithms, including G-CEALS itself.

For example, one of the datasets is the Contraceptive Method Choice dataset (OpenML ID 23). According to the OpenML description [42], "this dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of the interview. The problem is to predict the current contraceptive method choice (no use, long-term methods, or short-term methods) of a woman based on her demographic and socio-economic characteristics."

The dataset contains 1473 samples with 9 features: two numerical variables (age of the wife and number of children ever born), and seven categorical variables (wife's education, husband's education, wife's religion, wife's working status, husband's occupation, standard-of-living index, and media exposure). The cluster variable is the contraceptive method used, which takes three values: no use, long-term method, and short-term method.

Table 5.1 shows all of the datasets used. The following attributes and statistics are displayed: ID and name of the dataset, number of samples, number of all features, number of numerical features, number of categorical features, number of clusters, and feature-to-sample ratio (F-S ratio). The feature-to-sample ratio is the number of features divided by the number of samples and multiplied by 100. In these 16 sets, it ranges from 0.5 to 8.3, showcasing the variability between them, as a wide range highlights the robustness of the evaluation setup.

[1] F-S ratio = $100 \times$ #features / #samples

Table 5.1.: Overview of the datasets used in the experiments

OpenML ID	Name	Samples	Features	Numerical features	Categorical features	Clusters	F-S ratio[1]
11	Balance-scale	625	4	4	0	3	0.640
23	Cmc	1473	9	2	7	3	0.611
37	Diabetes	768	8	8	0	2	1.042
458	Analcatdata-authorship	841	70	69	1	4	8.323
469	Analcatdata-dmft	797	4	0	4	5	0.502
1049	Pc4	1458	37	37	0	2	2.538
1050	Pc3	1563	37	37	0	2	2.367
1063	Kc2	522	21	21	0	2	4.023
1067	Kc1	2109	21	21	0	2	0.996
1068	Pc1	1109	21	21	0	2	1.894
1464	Blood-transfusion	748	4	4	0	2	0.535
1480	Ilpd	583	10	9	1	2	1.715
1510	Wdbc	569	30	30	0	2	5.272
40975	Car	1728	6	0	6	4	0.347
40982	Steel-plates-fault	1941	27	27	0	7	1.391
40994	Climate-model	540	20	20	0	2	3.704

It is also worth noting that there are 11 datasets with only numerical features, 3 datasets with both numerical and categorical features, and the *40975* and *469* datasets have only categorical ones.

Table 5.2 presents the cluster distributions for each dataset, where each column corresponds to the relative frequency of one cluster. The majority of the datasets (10 out of 16) contain only two clusters, and in more than half of these cases the distributions are highly imbalanced, with the larger cluster accounting for over 80% of the samples.

Among the remaining datasets, dataset *469* exhibits an almost perfectly balanced distribution across five clusters, while dataset *40982* contains the largest number of clusters, namely seven. These observations highlight the variety in cluster structure across the datasets.

Table 5.2.: Cluster distributions

Dataset ID	C1	C2	C3	C4	C5	C6	C7
11	0.461	0.461	0.078	–	–	–	–
23	0.427	0.347	0.226	–	–	–	–
37	0.651	0.349	–	–	–	–	–
458	0.377	0.352	0.206	0.065	–	–	–
469	0.234	0.200	0.192	0.188	0.186	–	–
1049	0.878	0.122	–	–	–	–	–
1050	0.898	0.102	–	–	–	–	–
1063	0.795	0.205	–	–	–	–	–
1067	0.845	0.155	–	–	–	–	–
1068	0.931	0.069	–	–	–	–	–
1464	0.762	0.238	–	–	–	–	–
1480	0.714	0.286	–	–	–	–	–
1510	0.627	0.373	–	–	–	–	–
40975	0.700	0.222	0.040	0.038	–	–	–
40982	0.347	0.207	0.201	0.098	0.081	0.037	0.028
40994	0.915	0.085	–	–	–	–	–

5.2. Data Preprocessing

Although, as mentioned, the datasets were already clean and complete, and therefore required no extensive preparation, several preprocessing steps were necessary to adapt them to the model. These steps followed the same procedure as in TabDDPM.

Namely, categorical features were transformed using one-hot encoding, resulting in a set of binary variables for each feature. This ensures that categorical attributes can be handled by models that operate in a continuous feature space. This means that for the datasets with categorical variables, the number of features during the modeling part increases (cf. Table 5.3).

Table 5.3.: Number of features before and after one-hot encoding

Dataset ID	Features	Final features
23	9	24
458	70	81
469	4	21
1480	10	11
40975	6	21

The numerical features were standardized using a quantile transformation with the `QuantileTransformer` implementation from `scikit-learn`. Specifically, it maps the distribution of each feature to a target distribution, which is in our case, the standard normal distribution. This ensures that each transformed feature has Gaussian marginals,

which improves comparability between features with different scales. In the implementation a fixed `random_state = 0` was applied to ensure reproducibility.

Finally, the transformed categorical and numerical features were concatenated into a single processed dataset.[2]

Table 5.4.: Overview of the balanced datasets

ID	Name	Samples	Features	F-S ratio
37b	Diabetes-balanced	536	8	1.493
1049b	Pc4-balanced	356	37	10.393
1050b	Pc3-balanced	320	37	11.562
1063b	Kc2-balanced	214	21	9.813
1067b	Kc1-balanced	652	21	3.221
1068b	Pc1-balanced	154	21	13.636
1464b	Blood-transfusion-balanced	356	4	1.124
1480b	Ilpd-balanced	334	10	2.994
1510b	Wdbc-balanced	424	30	7.075
40994b	Climate-model-balanced	92	20	21.739

Also, since many of the two-cluster datasets are significantly imbalanced, balanced versions of them were created as well. Specifically, the majority class was undersampled by selecting samples uniformly at random until its size matched that of the minority class. An overview of these balanced versions is provided in Table 5.4.

5.3. Evaluation Metrics

The results are discussed via accuracy (ACC) and the adjusted Rand index (ARI) metrics.

Accuracy is calculated as

$$\mathrm{ACC} = \max_{m} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{y_{\mathrm{true}}(i) = m(y_{\mathrm{pred}}(i))\}\,, \tag{5.1}$$

where $y_{\mathrm{true}}(i)$ is the true label of sample i, and $y_{\mathrm{pred}}(i)$ is its predicted value. The function m is aligning the predicted and the true labels, which is found via the Hungarian algorithm [43].

ARI measures the similarity between the predicted cluster assignments, and the ground-truth assignments without matching the predicted and the real clusters. The Rand Index (RI) calculates the ratio of point pairs, which are either in the same cluster for both the predicted and true values (denoted by SS), or in different ones for both (denoted by DD). That is, if there are n points, then the Rand Index is defined as

$$\mathrm{RI} = \frac{\mathrm{SS} + \mathrm{DD}}{\binom{n}{2}}. \tag{5.2}$$

[2]The implementation of preprocessing steps is available on the GitHub repository for this thesis at `https://github.com/botond0401/CluTaD`.

The Adjusted Rand Index corrects RI by subtracting the expected RI ($\mathbb{E}[RI]$), i.e., the expected number of agreeing pairs if the clusterings were random, but their sizes were fixed. Then it is divided by the difference between the theoretical maximum of RI, which is trivially 1, and $\mathbb{E}[RI]$:

$$\text{ARI} = \frac{RI - \mathbb{E}[RI]}{1 - \mathbb{E}[RI]}. \tag{5.3}$$

ARI ranges from -1 to 1, where -1 is for the worst possible clustering assignments, 0 means random clustering assignments, e.g., predicting the same cluster for all points, and 1 is the perfect clustering.

5.4. Implementation details

To make **CluTaD-0** and **CluTaD** directly comparable with other deep clustering methods, we followed the same hyperparameter settings as in G-CEALS [5]. In that work, all methods use a fully connected AE with the architecture d-500-500-2000-m-2000-500-500-d, where d denotes the input dimension and m the latent feature's dimension. In more detail, the encoder maps the input d through three hidden layers down to m, while the decoder mirrors this, expanding back to d. For all experiments, the learning rate was set to 0.001 for the Adam optimizer, using a batch size of 256. Each method first pretrains the autoencoder for 1000 epochs and then fine-tunes with the final objective for another 1000 epochs.

The only difference in our setup is in the decoder: while the baseline methods use a 2000-500-500 structure, **CluTaD-0** and **CluTaD** adopt the TabDDPM-style decoder with a single hidden layer of size tuned between 500 and 1000.

The number of diffusion timesteps was set to $T = 100$, reflecting the relatively small dataset sizes. As in the baseline, we used a clustering loss weight of 0.1, and tuned the embedding dimension m over the same values $\{5, 10, 15, 20\}$.

6. Experimental Results

This chapter summarizes the results of the empirical evaluation of the proposed approach. We first report the clustering performance on the original benchmark datasets (cf. Chapter 5), and then investigate the phenomenon of cluster collapse in imbalanced datasets.

6.1. Accuracy and ARI on Original Datasets

As mentioned before, in the G-CEALS paper [5] the clustering accuracies and ARI scores were calculated on the 16 benchmark datasets for 12 clustering algorithms. Here, the same metrics for both **CluTaD-0** and **CluTaD** are shown in Table 6.1 and Table 6.2. The best results for each dataset are highlighted in bold. As in prior work, we calculate the average ranks and their deviations. Each algorithm receives a rank from 1 to 16 on each of the 16 datasets, with 1 assigned to the highest accuracy. In case of a tie, the next rank is skipped (e.g., two algorithms tied for first place, and the next best receives third place). The average rank is then calculated for each algorithm, and based on these averages we get an overall ranking from best to worst, 1 indicating the best.

Table 6.1.: Clustering accuracy (ACC) on the benchmark datasets

Dataset ID	*k*-means(X)	GMM(X)	*k*-means(Z)	GMM(Z)	DEC	IDEC	DEPICT	DynAE	DCN	AE-CM	DKM	G-CEALS	CluTaD-0	CluTaD
1063	80.3	80.3	68.0	72.8	67.0	66.9	66.1	79.5	80.3	79.6	79.6	**81.0**	79.5	79.5
40994	52.4	53.1	57.0	61.1	57.0	54.8	50.4	60.4	64.6	**91.5**	60.0	85.9	**91.5**	**91.5**
1510	91.2	94.0	92.1	89.5	91.4	93.1	93.7	58.0	77.7	62.7	80.2	**95.6**	95.3	**95.6**
1480	61.4	53.5	64.0	64.0	59.3	58.7	62.1	64.0	63.7	**71.3**	65.8	65.2	68.4	68.4
11	51.2	52.6	52.3	55.8	51.4	52.3	52.8	51.7	57.6	46.1	47.6	69.3	63.8	**74.6**
37	66.1	58.5	73.2	72.8	71.4	71.2	67.4	67.4	64.4	65.1	66.4	**74.1**	64.7	65.1
469	21.8	21.5	21.3	21.6	21.5	21.3	21.0	22.3	22.6	20.5	21.9	22.0	23.1	**23.2**
458	98.2	98.5	83.6	87.4	81.5	83.1	78.8	**99.9**	51.0	49.5	64.5	89.8	50.1	54.7
1464	68.3	57.6	73.9	69.1	66.4	66.4	56.1	66.0	74.1	**76.2**	64.3	74.1	76.2	76.2
1068	89.2	71.0	92.3	86.8	82.5	81.2	59.0	93.0	93.0	**93.1**	**93.1**	92.5	**93.1**	**93.1**
1049	81.9	72.2	84.3	72.2	70.0	69.3	66.5	70.5	87.1	87.9	87.5	85.3	**87.8**	**87.8**
23	42.4	39.6	42.0	43.4	**44.3**	43.8	42.0	38.8	40.7	42.7	36.9	44.2	43.0	43.7
1050	87.3	75.0	66.3	76.2	58.6	62.5	57.9	89.7	89.0	89.7	64.9	83.0	**89.8**	**89.8**
40975	31.2	31.2	31.1	34.7	31.1	31.1	38.3	46.1	56.5	70.0	36.1	49.2	**74.2**	70.0
40982	39.0	37.5	45.2	46.1	44.4	44.8	36.6	47.6	38.0	36.6	44.9	41.3	46.6	**50.6**
1067	83.3	72.6	**85.3**	77.0	80.1	79.3	62.1	77.3	84.7	84.7	**85.3**	85.2	84.5	84.5
Avg. rank	8.41	9.84	7.66	7.41	9.41	9.66	11.09	7.53	6.97	6.91	7.78	4.19	4.41	3.75
Std. rank	2.78	3.92	3.18	2.58	3.24	3.22	3.44	3.87	3.64	5.30	3.80	2.35	3.70	3.36
Overall rank	10	13	8	6	11	12	14	7	5	4	9	2	3	1

The results highlight several important points. Firstly, **CluTaD-0** achieved the third-best average rank, behind only G-CEALS and the full **CluTaD** model. This confirmed that the baseline design is already effective and served as a valuable sanity check, justifying the development of **CluTaD** itself.

CluTaD achieves the best overall accuracy rank, outperforming all baselines including G-CEALS. **CluTaD** is particularly strong on Dataset *11*, the Balance Scale dataset,

Table 6.2.: Adjusted Rand Index (ARI) on the benchmark datasets

Dataset ID	k-means(X)	GMM(X)	k-means(Z)	GMM(Z)	DEC	IDEC	DEPICT	DynAE	DCN	AE-CM	DKM	G-CEALS	CluTaD-0	CluTaD
1063	0.044	0.044	0.127	0.193	0.115	0.112	0.102	0.215	0.192	0.082	0.007	**0.335**	0.000	0.000
40994	0.001	0.003	0.017	**0.031**	0.017	0.009	-0.010	0.006	0.021	0.000	0.004	0.029	0.000	0.000
1510	0.677	0.774	0.707	0.620	0.684	0.744	0.762	0.020	0.295	0.000	0.347	**0.831**	0.818	**0.831**
1480	-0.072	-0.031	0.043	0.031	0.034	0.029	**0.048**	0.031	0.006	-0.001	-0.020	0.033	0.026	0.026
11	0.139	0.135	0.140	0.147	0.130	0.140	0.120	0.076	0.097	0.000	0.083	0.296	0.264	**0.419**
37	0.100	0.013	**0.211**	0.205	0.181	0.179	0.121	0.110	0.050	0.000	0.070	0.200	0.012	0.000
469	0.004	0.003	0.007	0.006	0.007	0.007	0.006	0.006	0.009	0.001	0.006	0.006	**0.014**	0.010
458	0.951	0.959	0.695	0.766	0.650	0.705	0.747	**0.996**	0.178	0.060	0.400	0.786	0.179	0.176
1464	0.034	-0.048	0.077	**0.126**	0.061	0.066	0.013	0.014	0.009	0.000	0.024	0.079	0.000	0.000
1068	**0.184**	0.060	0.157	0.176	0.105	0.110	0.025	0.030	0.080	0.000	0.014	0.148	0.000	0.000
1049	0.090	0.080	0.049	0.064	0.055	0.056	0.098	0.058	0.051	0.008	0.011	**0.105**	0.000	0.000
23	0.027	0.003	0.026	0.032	0.035	0.031	0.027	-0.007	0.018	0.000	-0.003	0.033	0.010	0.023
1050	0.046	**0.116**	-0.048	0.109	-0.020	-0.033	0.025	0.006	0.060	0.000	-0.057	0.073	0.000	0.000
40975	0.013	0.013	0.011	0.021	0.011	0.011	0.013	0.110	0.021	0.000	0.029	0.035	**0.277**	0.000
40982	0.156	0.142	0.217	0.207	0.218	0.221	0.125	**0.223**	0.082	0.007	0.192	0.163	0.165	0.167
1067	0.209	0.162	0.185	0.185	**0.218**	**0.218**	0.058	0.113	0.119	0.105	0.179	0.211	0.000	0.000
Avg. rank	7.41	7.22	6.69	5.59	6.63	6.50	8.06	7.34	8.19	10.34	9.06	4.91	8.34	8.72
Std. rank	3.19	3.28	2.56	2.51	1.95	1.84	2.28	3.28	3.06	2.69	2.63	2.27	3.85	3.52
Overall rank	8	6	5	2	4	3	9	7	10	14	13	1	11	12

which is based on weight and distance attributes with three clusters. Among the baseline methods the highest accuracy was achieved by G-CEALS at 69.3%, which **CluTaD** surpassed with 74.6%, meaning an increase of 5.3%. The ARI score also improved significantly from 0.296 to 0.419. The real and predicted clusters are visualized after applying t-SNE on the features in Figure 6.1. One can observe that t-SNE produces a well-separated group in the upper right corner, which does not correspond to any real cluster. However, **CluTaD** interprets it as a separate cluster, which negatively affects its accuracy. More interestingly, there are eight smaller groups of points in the main region. In reality, these belong to only two clusters, each consisting four of these groups. Although the four subgroups belonging to the same cluster are not positioned next to each other in the t-SNE space, **CluTaD** correctly assigns them to the same cluster, demonstrating its robustness.

A first weakness can be observed for Dataset *458*: while more than half of the methods achieve accuracies above 80% (with one reaching even 99.9%), both **CluTaD-0** and **CluTaD** achieve only 50.1% and 54.7%, respectively. This dataset contains 70 features, whereas all other cases involve fewer than 40, suggesting that **CluTaD** is not well-suited for datasets with a very high number of features.

To test this assumption, PCA was applied to reduce the dimensionality of Dataset *458* to four. On this reduced representation, **CluTaD** achieved 91.4% accuracy and an ARI of 0.786. While this is still below the best results (over 99%), it represents a substantial improvement compared to the original 70-dimensional case, supporting the hypothesis that the performance issue was due to the large number of features.

More importantly, ARI results reveal a notable weakness: **CluTaD** often has an ARI of 0, which means a collapse to a single dominant cluster. This still resulted in artificially high accuracy on several datasets, as they had an imbalanced two-cluster structure (e.g., Dataset *40994*, Dataset *1050*). This issue is further examined in one of the following sections (6.3).

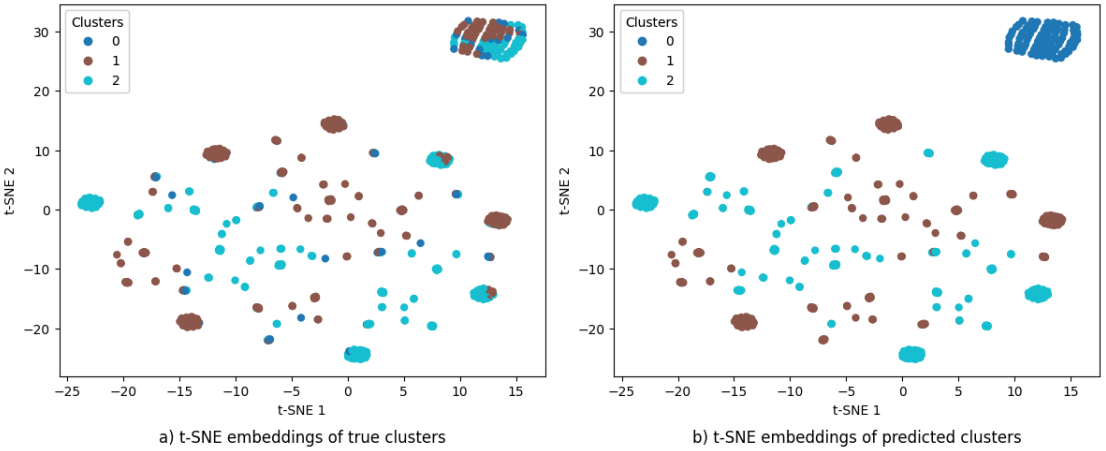

Figure 6.1.: Visualization of the clusters in Dataset *11*

6.2. Consistency Check of λ

To make our models comparable with the benchmark models, the λ parameter was fixed at 0.1 and not tuned. However, a consistency check was carried out. On five randomly selected datasets (*23*, *469*, *37*, *1464*, *1480*) we ran the **CluTaD** model for different λ values. The iteration ranged from the fixed value of 0.1 up to 1 (corresponding to equal clustering and reconstruction loss), in steps of 0.1. The accuracy and ARI results are shown in Figure 6.2.

These results show that 0.1 would have been the preferred option. Regarding accuracy, only for dataset *1480* is the highest value not at $\lambda = 0.1$. It is also worth noting that the runs appear more or less consistent, only dataset *1464* shows a large decrease after $\lambda = 0.1$, which suggests that the high value at 0.1 may have been a lucky anomaly, but this result also corresponds to a collapsed cluster. Regarding ARI, the model appears less consistent, with multiple jumps in almost all datasets. Again, most notably, dataset *1464* shows a sharp drop after the first iteration, while dataset *1480* shows an unusual peak at $\lambda = 0.7$, where ARI briefly increases to 0.06.

The results also indicate that cluster collapse cannot be solved simply by increasing this hyperparameter. Two of the datasets (*37* and *1464*) contain two clusters: in both cases, $\lambda = 0.1$ results in an ARI of 0, meaning a collapse into a single cluster. For dataset *1464*, the drop after the first iteration makes the results even worse, while for dataset *37*, accuracy decreases and ARI fluctuates slightly above and below zero.

To conclude, this consistency check confirms that the selected parameter value ($\lambda = 0.1$) is reasonable, since changing it does not lead to improvements in either accuracy or ARI.

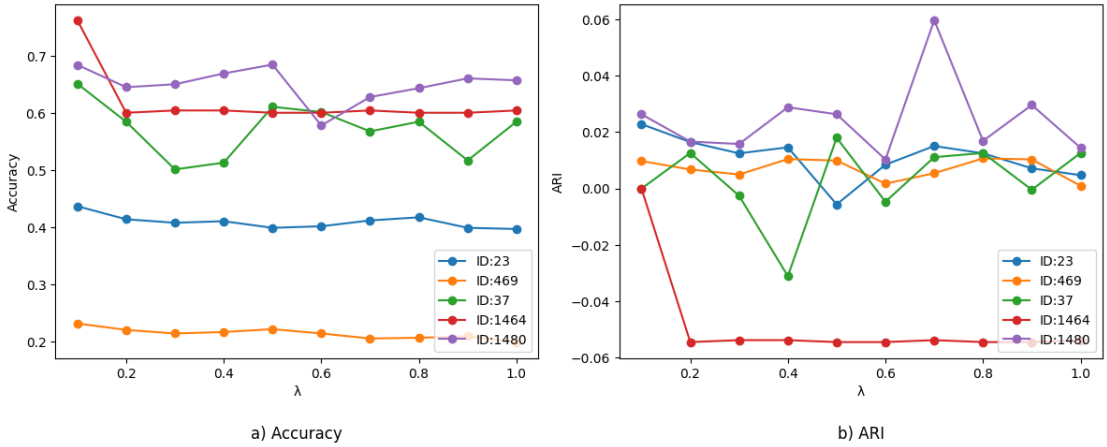

Figure 6.2.: Consistency check of λ

6.3. Balanced Versions of Imbalanced Datasets

CluTaD frequently merged all the data into one cluster on datasets with two clusters when one cluster dominated as the majority class (Datasets *37*, *1049*, *1050*, *1063*, *1067*, *1068*, *1464*, *40994*). This behavior also occurred for Dataset *40975*, despite it containing four classes. To evaluate if its weakness lies in data imbalance or in the low number of clusters, we investigated its performance on the same datasets by creating balanced versions.

Balanced datasets were created by subsampling the majority class to match the number of samples from the minority class. In these experiments, we also re-enabled the early stopping criterion based on cluster weights, which previously had been disabled for two-cluster datasets.

We compared **CluTaD** with two classical methods, k-means and GMM, on the balanced datasets using 10 random seeds and reporting the best accuracy for each baseline. Results are shown in Table 6.3 and Table 6.4. Balanced dataset identifiers are denoted with an additional b. Since there were deep clustering benchmarks in this setting, we were not restricted to fixed hyperparameters. Therefore, we also tuned the λ hyperparameter between 0.1 and 1.

Our findings are summarized as follows:

- **CluTaD** no longer collapses to trivial one-cluster solutions.
- On most balanced datasets, **CluTaD** outperformed k-means and GMM, demonstrating that diffusion-based clustering can indeed produce meaningful partitions even in two-cluster settings.
- The reintroduction of early stopping prevented degenerate cluster weights and stabilized training.

Table 6.3.: Clustering accuracy (ACC) on the balanced datasets

Dataset ID	k-means	GMM	CluTaD
1480b	0.521	0.521	**0.569**
37b	0.511	0.519	**0.554**
1067b	0.670	0.664	**0.673**
1063b	0.715	0.706	**0.724**
1068b	0.643	0.610	**0.656**
1050b	0.603	0.603	**0.638**
1049b	0.711	**0.733**	0.666
1464b	0.548	0.548	**0.551**
1510b	**0.913**	0.899	0.899
40994b	0.500	0.500	**0.685**

Table 6.4.: Adjusted Rand Index (ARI) on the balanced datasets

Dataset ID	k-means	GMM	CluTaD
1480b	0.001	0.001	**0.016**
37b	-0.001	0.000	**0.011**
1067b	0.115	0.106	**0.119**
1063b	0.181	0.166	**0.197**
1068b	0.076	0.043	**0.091**
1050b	0.041	0.041	**0.073**
1049b	0.175	**0.215**	0.108
1464b	0.008	0.008	**0.009**
1510b	**0.681**	0.635	0.635
40994b	-0.011	-0.011	**0.127**

- For Datasets *1049* and *1510*, traditional clustering algorithms performed better than **CluTaD**.
- The most notable improvement occurred on Dataset *40994*, where baseline methods achieved only random-level accuracy (50%), while **CluTaD** reached 68.5% accuracy and an ARI of 0.127.

6.4. Discussion

The experimental results provided evidence that diffusion-based deep clustering is a promising approach when clustering tabular data, but also highlighted important limitations.

First, the results on 16 benchmark datasets indicate that both **CluTaD-0** and **CluTaD** achieve competitive performance compared to other state-of-the-art baselines in terms of accuracy. **CluTaD-0** ranked third overall. The full **CluTaD** model achieved the best average accuracy rank across all methods, surpassing even G-CEALS. The most

significant example is Dataset *11*, where **CluTaD** improved accuracy by 5.3 percentage points and ARI from 0.296 to 0.419. These results prove that diffusion models can indeed successfully capture the structure of heterogeneous tabular data.

Second, the evaluation also revealed several weaknesses. For Dataset *458*, both **CluTaD-0** and **CluTaD** underperformed dramatically, with accuracies near 50%, while most baselines exceeded 80%. The main characteristic of this dataset is its high dimensionality (70 features), suggesting that **CluTaD** is not effective when applied to very high-dimensional data.

A more general limitation is exposed by the ARI scores: in multiple datasets with two highly imbalanced clusters, **CluTaD** frequently collapsed into a single cluster, leading to ARI values of 0. Thus, experiments were run on balanced versions of the imbalanced datasets and it was confirmed that these issues are not inherent to the model architecture. Once the class distributions are equalized and early stopping based on the cluster prior weights was reintroduced, **CluTaD** consistently outperformed k-means and GMM, except for 2 datasets, where traditional clustering remained superior. This confirms that the observed failures on imbalanced datasets are not due to an inability of the model to separate clusters when there are only two of them, but rather to its sensitivity to significant imbalance. We also observed that reweighting the loss function resulted in multiple clusters but reduced overall performance, making it an unsuitable solution. These findings suggest that **CluTaD** is best suited for datasets with moderate imbalance, whereas extreme cases may require specialized reweighting techniques or alternative clustering paradigms.

Overall, **CluTaD** demonstrates strong potential, with competitive performance compared to established methods on several diverse tabular datasets. At the same time, the observed limitations point to potential directions for future work: incorporating an explicit workflow to handle imbalanced clusters or high-dimensionality and exploring more robust clustering objectives to mitigate cluster collapse. Addressing these would further strengthen the applicability of diffusion-based deep clustering methods in practical tabular learning scenarios.

7. Conclusion

In this work, I have investigated the usage of diffusion models for deep clustering tabular data, evaluating whether they can achieve competitive clustering performance despite the well-known challenges of using deep learning on tabular datasets. A literature review revealed that no prior work has combined deep clustering, tabular data, and diffusion models, and that even their pairwise combinations have been explored only minimally. However, existing models demonstrate competitive performance in each of these individual pairings. ClusterDDPM is a competitive clustering algorithm utilizing the diffusion model architecture for mainly image data, TabDDPM is a well-established diffusion model type designed for tabular data generation, and G-CEALS is a state-of-the-art deep clustering algorithm for tabular data. Based on this observation I made the hypothesis that combining the central building blocks of these models would enable the development of a clustering approach well suited to the challenges of tabular data.

Before implementing the final model, I first combined ClusterDDPM and TabDDPM to examine whether the research direction was reasonable. Evaluation on the 16 benchmark tabular datasets demonstrated that **CluTaD-0** outperformed all but one of the baseline clustering algorithms. This directly led to the creation of **CluTaD**, which incorporated the insights of G-CEALS into the **CluTaD-0** model. I also tested this model on the same datasets, and it outperformed all competitor models, including G-CEALS itself on several datasets. It is important to note that it yielded poor results on high-dimensional data, and that for highly imbalanced two-cluster datasets it produced trivial, one-cluster solutions. I reran the model on balanced versions of these datasets, where it was able to generate competitive results, indicating that it can suffer from imbalance in a dataset, but is not directly affected by the two-cluster structure itself.

I was able to address the research questions as detailed below:

- **Is the TabDDPM architecture, originally designed for imputation and synthetic data generation, also effective for clustering tasks?** The experiments with **CluTaD-0** confirmed that TabDDPM's denoising process for both numerical and categorical data are highly effective for clustering tabular data. Its ability to handle mixed features provided an advantage over other deep clustering models. Although TabDDPM alone was not designed for clustering objectives, when combined with clustering losses it resulted in strong results validating its applicability beyond just imputation and data generation.

- **How does the proposed CluTaD model compare to other deep clustering approaches in terms of clustering accuracy on benchmark tabular datasets?** The empirical evaluation demonstrated that both **CluTaD-0** and the full

CluTaD model achieve competitive or superior clustering accuracy compared to established baselines, including the G-CEALS model. **CluTaD-0** already outperformed almost all other methods on the tested benchmark datasets, and the full **CluTaD** model consistently improved compared to **CluTaD-0**, surpassing even G-CEALS in terms of average rank.

- **Does the proposed method exhibit particular strengths or weaknesses under specific dataset conditions?** The proposed method shows strong performance on most benchmark datasets, although no specific dataset conditions were identified as particularly favourable. In contrast, it exhibits clear weaknesses under certain challenging conditions. It struggles with high-dimensional data and with strongly imbalanced two-cluster datasets, as it might collapse to trivial solutions.

Overall, these findings provide favourable answers to all three of my research questions, while also highlighting some limitations, most notably the challenges posed by dimensionality and class imbalance, which should be addressed in future work.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised Deep Embedding for Clustering Analysis," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 478–487.

[3] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *Proceedings of the International Conference on Machine Learning*, 2017, pp. 3861–3870.

[4] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," *Proceedings of Internationl Joint Conference on Artificial Intelligence*, vol. 17, pp. 1753–1759, 2017.

[5] S. B. Rabbani, I. V. Medri, and M. D. Samad, "Deep clustering of tabular data by weighted gaussian distribution learning," *Neurocomputing*, vol. 623, p. 128537, 2025.

[6] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in neural information processing systems*, vol. 34, pp. 18 932–18 943, 2021.

[7] S. Abrar, A. Sekmen, and M. D. Samad, "Effectiveness of deep image embedding clustering methods on tabular data," in *Proceedings of the International Conference on Advanced Computational Intelligence*, 2023, pp. 1–7.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[9] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020.

[10] R. Yang, P. Hu, X. Peng, X. Liu, and Y. Li, "Dific: Your diffusion model holds the secret to fine-grained clustering," *arXiv preprint arXiv:2412.18838*, 2024.

[11] J. Yan, J. Liu, and Z.-Y. Zhang, "ClusterDDPM: An EM clustering framework with denoising diffusion probabilistic models," *Information Sciences*, vol. 720, p. 122518, 2025.

[12] A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko, "TabDDPM: Modelling tabular data with diffusion models," in *Proceedings of the International Conference on Machine Learning*, 2023, pp. 17 564–17 579.

[13] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.

[14] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[15] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1–3, pp. 37–52, 1987.

[16] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[17] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[18] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[19] Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu, and L. He, "Deep clustering: A comprehensive survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 4, pp. 5858–5878, 2025.

[20] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[21] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[22] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*, 2015, pp. 2256–2265.

[23] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of the International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[24] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society: series B (methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[25] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, "Argmax flows and multinomial diffusion: Learning categorical distributions," *Advances in neural information processing systems*, vol. 34, pp. 12 454–12 465, 2021.

[26] R. Quinlan, "Thyroid Disease," UCI Machine Learning Repository, 1986, DOI: https://doi.org/10.24432/C5D010.

[27] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *2015 IEEE symposium series on computational intelligence*. IEEE, 2015, pp. 159–166.

[28] OpenML, "OpenML: Open machine learning," https://www.openml.org, 2025, accessed: 2025-04-15.

[29] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[32] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka, "Well-tuned simple nets excel on tabular datasets," *Proceedings of Advances in neural information processing systems*, vol. 34, pp. 23 928–23 941, 2021.

[33] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 6, pp. 7499–7519, 2024.

[34] H. T. Rauf, A. Freitas, and N. W. Paton, "TableDC: Deep clustering for tabular data," *arXiv preprint arXiv:2405.17723*, 2024.

[35] J. Svirsky and O. Lindenbaum, "Interpretable deep clustering for tabular data," in *Proceedings of the International Conference on Machine Learning*, 2024, pp. 47 314–47 330.

[36] S. Zheng and N. Charoenphakdee, "Diffusion models for missing value imputation in tabular data," *arXiv preprint arXiv:2210.17128*, 2022.

[37] M. Villaizán-Vallelado, M. Salvatori, C. Segura, and I. Arapakis, "Diffusion models for tabular data imputation and synthetic data generation," *arXiv preprint arXiv:2407.02549*, 2024.

[38] T. Sattarov, M. Schreyer, and D. Borth, "FinDiff: Diffusion models for financial tabular data generation," in *Proceedings of the Fourth ACM International Conference on AI in Finance*, 2023, pp. 64–72.

[39] K. Preechakul, N. Chatthee, S. Wizadwongsa, and S. Suwajanakorn, "Diffusion autoencoders: Toward a meaningful and decodable representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 619–10 629.

[40] Y. Kim, K. Lee, M. Park, B. Na, and I.-C. Moon, "Diffusion bridge autoencoders for unsupervised representation learning," *arXiv preprint arXiv:2405.17111*, 2024.

[41] J. d. S. Goncalves, L. Manduchi, M. Vandenhirtz, and J. E. Vogt, "Structured generations: Using hierarchical clusters to guide diffusion models," *arXiv preprint arXiv:2407.06124*, 2024.

[42] OpenML, "Contraceptive method choice (dataset id 23)," https://www.openml.org/d/23, 2025, accessed: 2025-04-15.

[43] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

Acronyms

AE Autoencoder. 7, 11, 18, 25, 27, 43

ARI Adjusted Rand Index. 42, 43, 45–47, 49, 50

DAE Denoising Autoencoder. 23

DC Deep Clustering. 1, 6, 7, 18, 19, 23, 26

DDPM Denoising Diffusion Probabilistic Model. 2, 11, 12, 15, 29, 38

DNN Deep Neural Network. 6

GAN Generative Adversarial Network. 13

GMM Gaussian Mixture Model. 12, 13, 32–35, 38, 48, 50, 60

MLP Multilayer Perceptron. 20, 34

NN Neural Network. 1

PCA Principal Component Analysis. 46

RI Rand Index. 43

VAE Variational Autoencoder. 13

A. Appendix

A.1. TabDDPM Reconstruction Validation

This section presents supplementary validation experiments confirming that TabDDPM can indeed reconstruct data distributions from their noised versions.

To verify this, TabDDPM was run on the OpenML dataset *1480*, which has one categorical and nine numerical features, using 1000 diffusion steps. At every 100 steps, as well as at the first step, the reconstructed distributions of both categorical and numerical variables were compared with the original dataset (see Figure A.1).

The results show that:

- For categorical features, the reconstructed distributions converge rapidly to the original ones. The empirical histograms of the generated and true categories are essentially identical.
- For numerical features, the reconstructed densities progressively approach the original distribution as the diffusion process proceeds. Small discrepancies remain, but the alignment becomes increasingly accurate.

These observations provide evidence that TabDDPM is not only a model for generating synthetic tabular data but also reliable when it comes to reconstructing tabular data distributions after noising them. This capability justifies its role within our framework, where accurate distribution recovery is crucial for guiding latent representation learning.

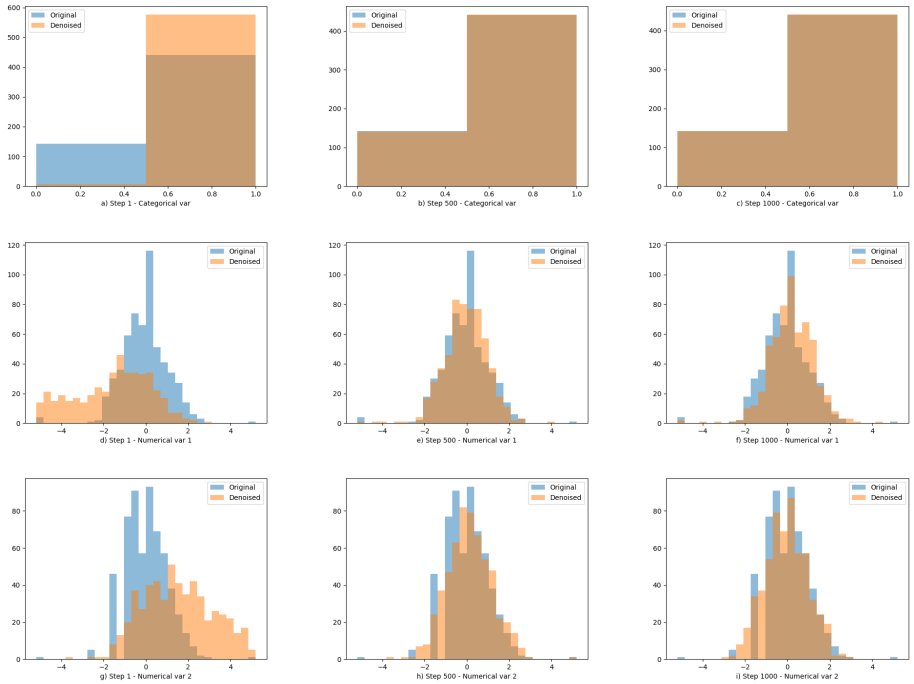

Figure A.1.: Validation of TabDDPM's reconstruction ability on dataset *1480*. (a–c) For the categorical feature, distributions at steps 1, 500, and 1000 show progressive alignment. (d–f) Same pattern for numerical feature 1. (g–i) Same pattern for numerical feature 2.

A.2. Baseline Model: CluTaD-0

This section details the baseline version of **CluTaD**, including its architecture, loss formulation, and training procedure.

CluTaD-0 shares the same structural foundation as **CluTaD** but without the modifications inherited from G-CEALS. That means it is constructed by modifying ClusterDDPM to use the heterogeneous forward and reverse processes from TabDDPM. While it keeps the overall EM-style optimization structure and Gaussian latent space of ClusterDDPM, it replaces the original continuous-only diffusion mechanism with the tabular-friendly diffusion scheme introduced in TabDDPM. This means that it also follows the EM-framework, which is as follows:

- **E-step:** Encode samples and fit a GMM in latent space to update $\boldsymbol{\pi}$, $\boldsymbol{M}$, and $\boldsymbol{\Sigma}$.
- **M-step:** Train encoder and denoiser to minimize the reconstruction and clustering losses.

While its E-step is the exact same one as that of **CluTaD** (see in Figure 4.1), in the M-step there is a major distinction in how the clustering loss is calculated, as ClusterDDPM's

loss function is used, the changes from G-CEALS are not added (see in Figure A.2). Algorithm 5 summarizes the steps. Empirical evaluation and results are presented in Chapter 6.

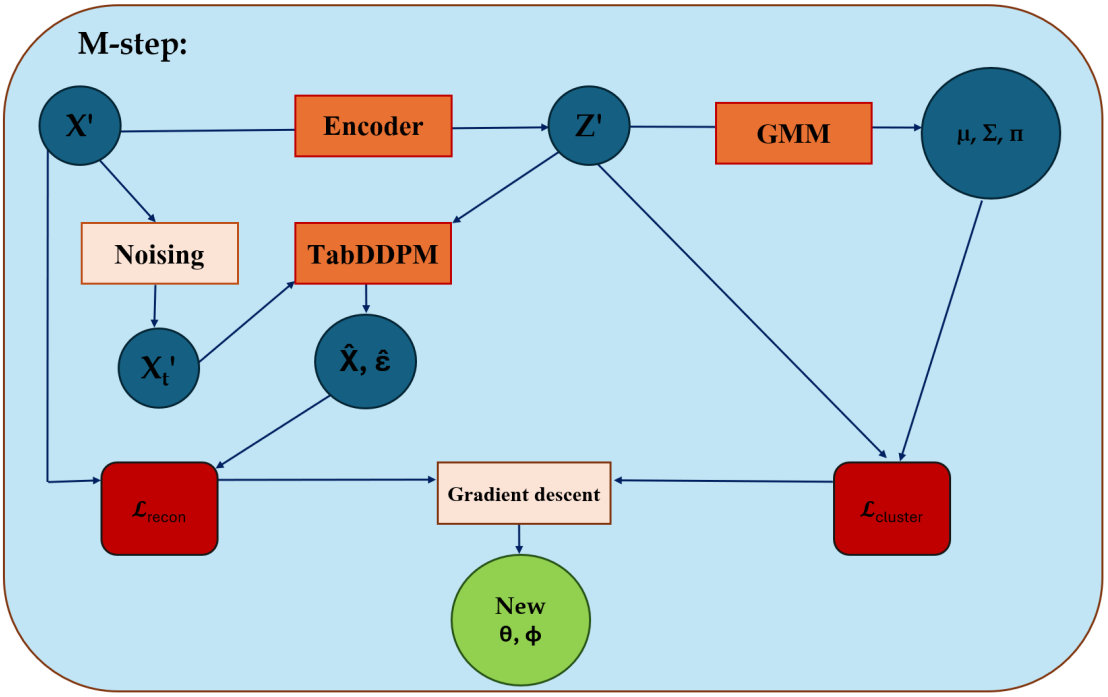

Figure A.2.: M-step of **CluTaD-0**

Algorithm 5: CluTaD-0

Input: Dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$, $\mathbf{x}^{(i)} \in \mathbb{R}^d$, number of clusters k, mini-batch size, and number of epochs num_epochs

Output: Soft cluster assignments

Initialize encoder $g_{\boldsymbol{\phi}}$ and TabDDPM $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$;

for *epoch* $= 1, \ldots,$ *num_ epochs* **do**

`// E-step: Clustering in latent space`

Encode samples: $\mathbf{Z} \leftarrow g_{\boldsymbol{\phi}}(\mathbf{X})$;

Fit a GMM with k components on $\mathbf{Z}$;

`// M-step: Conditional DDPM and encoder training`

for *each mini-batch of indices* $\mathcal{B} \subset \{1, \ldots, n\}$ **do**

for *all* $i \in \mathcal{B}$ **do**

$\mathbf{z}^{(i)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)})$;

Sample time step t and noise $\boldsymbol{\epsilon}$;

Apply heterogeneous diffusion to obtain $\mathbf{x}_t^{(i)}$;

Get TabDDPM predictions:

$$\left(\hat{\boldsymbol{\epsilon}}_i,\ \hat{\mathbf{x}}_{0,1}^{(i)}, \ldots, \hat{\mathbf{x}}_{0,C}^{(i)}\right) \leftarrow \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left([\mathbf{x}_t^{(i)}, \mathbf{z}^{(i)}], t\right);$$

Compute reconstruction loss:

$$\mathcal{L}_{\text{recon}} = \sum_{i \in \mathcal{B}} \left(\underbrace{\|\hat{\boldsymbol{\epsilon}}_i - \boldsymbol{\epsilon}_i\|^2}_{\text{MSE}} + \underbrace{\frac{1}{C} \sum_{c=1}^{C} \text{KL}\big(\mathbf{x}_{0,c}^{(i)} \,\|\, \hat{\mathbf{x}}_{0,c}^{(i)}\big)}_{\text{Categorical KL divergence}} \right);$$

Compute clustering loss:

$$\mathcal{L}_{\text{cluster}} = \sum_{i \in \mathcal{B}} \text{KL}\big(q(\mathbf{z}^{(i)}, \boldsymbol{\pi} \mid \mathbf{x}^{(i)}) \,\|\, p_{\text{GMM}}(\mathbf{z}^{(i)}, \boldsymbol{\pi})\big)$$

Compute batch loss: $\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda \cdot \mathcal{L}_{\text{cluster}}$;

Update parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ via gradient descent;

return *Final cluster assignments from GMM on* $g_{\boldsymbol{\phi}}(\mathbf{X})$

Trainable parameters for CluTaD-0:

- Encoder network parameters: $\boldsymbol{\phi}$
- Denoising network parameters: $\boldsymbol{\theta}$
- Cluster centroids: $\boldsymbol{M} = \{\boldsymbol{\mu}_j\}_{j=1}^{k}$
- Cluster covariance matrices: $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_j\}_{j=1}^{k}$
- Cluster weights: $\boldsymbol{\pi} = \{\pi_j\}_{j=1}^{k}$

A.3. Documentation of AI Tool Usage

Table A.1.: Documentation of AI tool usage

AI Tool	**Period of Use**	**Type of Use**
DeepL	December 2025	Used to translate the abstract. Final review and approval of the translation were done by me.
ChatGPT	September–December 2025	Used for linguistic support (grammar, style, clarity). No substantial content was generated without my review and revision.
ChatGPT	May–September 2025	Used for assistance with coding (debugging, analyzing error messages, and suggesting implementations). All final decisions regarding code and algorithms were made by me.