Transformers Can Do Bayesian Clustering

by

Prajit Bhaskaran

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday July 4, 2025 at 11:00 AM.

Student number:	5025397	
Project duration:	November 15, 2024 – July 4, 2025	
Thesis committee:	Prof. dr. M. J. T. Reinders,	TU Delft, chair
	Assistant Prof. dr. T. J. Viering,	TU Delft, supervisor
	Assistant Prof. dr. W. Böhmer,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

Acknowledgments

I would like to begin by sincerely thanking Dr. Tom Viering for his unwavering support and patience as a mentor throughout my academic journey. Our discussions have greatly helped me develop a more critical and thoughtful approach to conducting research.

I am also grateful to Ojas Shirekar for his valuable support and insightful contributions during the course of my thesis. Your ideas and observations have been very helpful in shaping the direction of this research.

Finally, I would like to express my heartfelt appreciation to my family and friends for their constant encouragement and unconditional support throughout this project.

Abstract

Motivation: Clustering is an unsupervised learning task with broad applications. Traditional clustering methods often rely on point estimates of model parameters, which can limit their ability to capture uncertainty. Bayesian clustering addresses this by incorporating uncertainty into parameter estimation. However, existing Bayesian inference methods like Markov Chain Monte Carlo and Variational Inference are computationally intensive and can produce biased approximations. To overcome these limitations, we propose Cluster-PFN, a transformer-based model inspired by Prior-Data Fitted Networks [30]. Cluster-PFN simultaneously approximates the posterior distributions over the cluster assignments of individual data points and the total number of clusters for the given dataset in a single forward pass. Our model provides fast and accurate Bayesian clustering, supporting data with up to five features and conditioning on user-specified cluster counts.

Results: Our results demonstrate that Cluster-PFN can predict the number of clusters up to 20% more accurately than standard heuristics. It also outperforms the Bayesian Gaussian Mixture Model using Variational Inference (Bayesian GMM VI), achieving up to 60% higher scores on certain external metrics while being up to 20 times faster during inference. Additionally, Cluster-PFN surpasses both the traditional Gaussian Mixture Model and K-means++ across the same external evaluation metrics.

Code Availability: Our code is available at our GitHub repository .

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
DL	Deep Learning
EM	Expectation-Maximization
MLE	Maximum Likelihood Estimate
MCMC	Markov Chain Monte Carlo
VI	Variational Inference
PFN	Prior-Data Fitted Network
GMM	Gaussian Mixture Model
ICL	In-context Learning
PPD	Posterior Predictive Distribution
MLP	Multi-layer Perceptron
ARI	Adjusted Rand Index
AMI	Adjusted Mutual Information
NLL	Negative Log-likelihood
AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion

1

Introduction

Clustering is a common form of unsupervised learning that aims to find similar data points that can be grouped together. Clustering has many applications such as customer segmentation in marketing and retail [26], anomaly detection in healthcare [2, 40], identifying suspicious financial transactions in the banking industry [43], and grouping similar highway hubs in urban planning [29].

Clustering data is a challenging task due to the absence of labels. Numerous clustering algorithms exist, each utilizing different techniques to cluster data. One approach is model-based clustering which uses probabilistic mixture models where each mixture component corresponds to a particular cluster [18, 20]. To compute the optimal parameters of the model, an iterative procedure called the Expectation-Maximization (EM) algorithm is typically used. Once the maximum likelihood estimates (MLEs) of the parameters are obtained, cluster assignments for individual data points can be determined. While mixture models provide probabilities for data point assignments, they do not account for uncertainty in the parameter estimates themselves.

One way to address this challenge is through Bayesian clustering. Unlike traditional clustering methods, which rely on point estimates for model parameters, Bayesian clustering incorporates uncertainty into these parameters, resulting in more robust and uncertainty-aware predictions. By placing prior distributions on the model parameters, the clustering model can update its beliefs based on observed data. However, Bayesian clustering models have a few drawbacks. The inference algorithms commonly used, such as Markov Chain Monte Carlo (MCMC) and Variational Inference (VI), are often computationally expensive [7, 31] and can introduce biased approximations [9] when estimating posterior distributions.

Problem Statement: Given a dataset $X = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ and corresponding latent variables $y_i \in \{0, \ldots, K-1\}$, where K (the number of clusters) is unknown, the objective is to estimate the posterior distribution over cluster assignments $p(y_i \mid X)$ for each data point, as well as the posterior over the number of clusters $p(K \mid X)$. In addition, the model should

support user-specified conditions on the number of cluster assignments. That is, given a query for a particular value of K, the model should produce cluster assignments conditioned on that value, effectively estimating $p(y_i \mid X, K)$.

Müller et al. [30] introduced the Prior-Data Fitted Network (PFN), a deep learning model that approximates posterior distributions for supervised tasks in a single forward pass. Their results demonstrate that PFNs can outperform traditional Bayesian inference methods in both speed and accuracy.

Motivated by this, we aim to extend the PFN framework to the unsupervised setting of clustering. Specifically, we propose Cluster-PFN, a transformer-based model that performs Bayesian clustering. We demonstrate that:

- Cluster-PFN can cluster data with up to five features, providing posterior probabilities for each data point's cluster membership.
- Cluster-PFN can adapt its clustering behavior conditioned on a user-specified number of clusters.
- Cluster-PFN predicts the number of clusters up to 20% more accurately than standard heuristics such as the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and silhouette score.
- Cluster-PFN outperforms the Bayesian Gaussian Mixture Model with Variational Inference approximation (Bayesian GMM VI), achieving up to 60% better scores on external clustering metrics—Adjusted Rand Index, Adjusted Mutual Information, and Negative Log-Likelihood—while also running an order of magnitude faster (approximately 20×) in terms of computational efficiency.
- Cluster-PFN also surpasses the Gaussian Mixture Model and K-means++ on external clustering metrics—Adjusted Rand Index, Adjusted Mutual Information, and Purity—achieving improvements of up to 50%.

This thesis is structured as follows: Chapter 2 outlines the relevant background information and related works that discuss Bayesian statistics and clustering, transformers, and Prior-Data Fitted Networks. Chapter 3 presents the objectives, architecture, and research questions of the Cluster-PFN. Chapter 4 highlights the experimental setup which delves into the data generation, and the metrics that we use to evaluate our model. Chapter 5 showcases the experiments and results of the models. Chapter 6 discusses the findings and outlines future work in this area, while Chapter 7 presents the conclusions of this research.

2

Background and Related Work

This chapter outlines a high-level overview of the context in which this thesis was written and provides the preliminary knowledge to understand the main matter.

2.1. Bayesian Statistics and Clustering Applications

Traditional machine learning methods estimate parameters based solely on the observed data, however, these kinds of approaches ignore uncertainty in the parameter estimates. Bayesian statistics addresses this limitation by placing probability distributions over the parameters, thereby modeling uncertainty explicitly. This process is known as inference [31]. To represent uncertainty, we begin by defining a prior distribution $P(\theta)$ which captures our beliefs about the model parameters, θ, before observing any data. When data x is observed, we update these beliefs using the likelihood $P(x|\theta)$, which represents the probability of observing the data given specific parameter values. The marginal likelihood (also known as the evidence), denoted $P(x)$ represents the total probability of the data under all possible parameter values. Using Bayes' rule, we obtain the posterior distribution $P(\theta|x)$ which combines the prior and likelihood to reflect our updated beliefs about the parameters after observing the data:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} = \frac{P(x|\theta)P(\theta)}{\int_{\theta'} P(\theta')P(x|\theta')\,d\theta'} \tag{2.1}$$

We can rewrite the marginal likelihood as an integral by marginalizing over our parameter θ. Once we have the posterior distribution, we can compute the posterior predictive distribution (PPD).

$$P(z_n|x, x_n) = \int_{\theta} P(z_n|x_n, \theta)P(\theta|x)\,d\theta \tag{2.2}$$

Here, x_n refers to a new data sample with unknown output and z_n represents the distribution of the output, while x refers to the observed data. The purpose of equation 2.2 is to show that

predictions are made by integrating over an infinite set of parameter values, weighted by the posterior distribution. This results in a complete distribution over y_n, accounting for all possible models- a process known as Bayesian Model Averaging.

Referring back to equation 2.1, computing the marginal likelihood for non-trivial tasks is difficult and often intractable. This intractability arises due to the high dimensionality of the parameter space θ, the complexity of the likelihood function which makes integration challenging, and the lack of closed-form solutions.

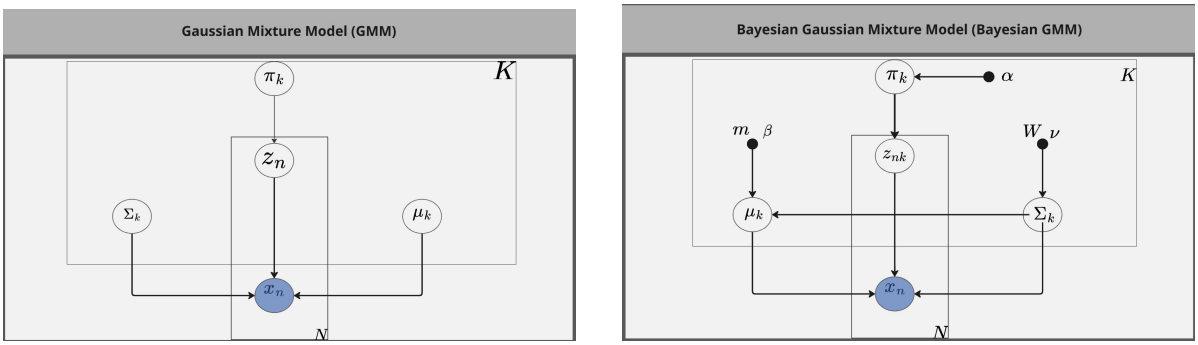
To address this, various approximation methods have been developed to estimate the posterior. The two most prominent methods are Markov Chain Monte Carlo (MCMC) [4, 35, 42], and Variational Inference (VI) [23, 27, 41]. In this work, we will focus on VI.

2.1.1. Traditional Clustering

In this section, we define the K-means++ and GMM clustering algorithms.

K-means++ is an extension of the K-means algorithm by selecting initial centroids more strategically and works as follows:

1. Randomly select the first centroid from the data points.
2. For each remaining point, compute the distance to the nearest chosen centroid.
3. Select the next centroid with a probability proportional to the square of the distance.
4. Repeat until k centroids are chosen.



(a) Gaussian Mixture Model **(b)** Bayesian Gaussian Mixture Model

Figure 2.1: Plate diagrams for standard GMM (left) and Bayesian GMM (right).

While K-means++ is a simple, distance-based clustering method that partitions data into k clusters, it does not provide a probabilistic interpretation or account for uncertainty in cluster assignments.

We now turn to the Gaussian Mixture Model (GMM), which offers a probabilistic framework for clustering. Unlike K-means++, GMM models the data as a mixture of Gaussian distributions and allows for soft cluster assignments, where each data point has a probability of belonging to each cluster. The GMM also forms the basis for Bayesian clustering methods and the application of VI.

The GMM assumes data is generated from the process illustrated in 2.1a which presents the

plate diagram for a GMM. In the diagram, each observed data point x_n is generated by first selecting a cluster k according to the categorical distribution defined by π_k, and then sampling from the corresponding Gaussian distribution $N(u_k, \Sigma_k)$. The label given to the data point is then z_k. The plate notation also indicates repetition where the outer plate corresponds to the N data points, and the inner plates denote the K components, each with its associated parameters π, μ, Σ. The marginal likelihood of a data point x_n can be written as:

$$p(x_n|\pi, \mu, \Sigma) = \sum_{k=1}^{K} \pi_k \, N(x_n|u_k, \Sigma_k) \tag{2.3}$$

Extending this to the entire dataset $x = \{x_1, x_2...x_n\}$ the likelihood of the data under the GMM is:

$$p(x|\pi, \mu, \Sigma) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k) \tag{2.4}$$

Taking the logarithm of the data likelihood yields the log-likelihood function:

$$\ln p(x|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k) \right\} \tag{2.5}$$

We cannot directly maximize this likelihood because as our plate diagram displays, the cluster assignments z_n are hidden and not observed by the data. Instead, we must use the joint-likelihood representation. We introduce a binary variable z using a one-hot representation, where a particular element $z_k = 1$ if a data point belongs to cluster k, and $z_k = 0$ otherwise. Extending this to all data points, let z_{nk} denote the cluster assignment of data point x_n, such that $z_{nk} = 1$ if x_n is assigned to cluster k, and $z_{nk} = 0$ otherwise. Using this representation, the joint log-likelihood can be written as:

$$\log p(x, z|\pi, \mu, \Sigma) = \log \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} N(x_n|\mu_k, \Sigma_K)^{z_{nk}} = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} (\log \pi_k + \log N(x_n|\mu_k, \Sigma_k)) \tag{2.6}$$

while the posterior $p(z_{nk} = 1|x_n)$ can be written as:

$$p(z_{nk} = 1|x_n, \pi, \mu, \Sigma) = \frac{p(z_{nk} = 1)p(x_n|z_{nk} = 1)}{\sum_{j=1}^{K} p(z_{nj} = 1)p(x_n|z_{nj} = 1)} = \frac{\pi_k N(\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(\mu_j, \Sigma_j)} = r_{nk} \tag{2.7}$$

$p(z_{nk} = 1|x_n, \pi, \mu, \Sigma)$ can also be viewed as the responsibility that component k takes for 'explaining' a particular observation x_n.

This formulation shown in equation 2.6 is now easier to optimize because we see that if z_{nk} were known, analytical solutions can be found for parameters π, μ, and Σ by taking derivatives. We see that the posterior over our latent cluster assignments depends on the parameters π, μ, and Σ, and our parameters depend on the posterior of z. We therefore introduce the iterative optimization algorithm Expectation-Maximization (EM). The EM algorithm consists of the E-step and the M-step. In the E-step, we estimate the expected values of the hidden variables z_{nk} given the current parameters. In the M-step, we maximize the expected complete-data log-likelihood using the previously computed responsibilities. The EM algorithm for the GMM case can be formulated in algorithm 1.

Algorithm 1 Expectation-Maximization (EM) Algorithm for GMM

Initialize: Parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$
repeat
  E-step: Compute responsibilities:
  $r_{nk} = \dfrac{\pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n \mid \mu_j, \Sigma_j)}$
  M-step: Update parameters:
  $N_k = \sum_{n=1}^{N} r_{nk}, \quad \pi_k = \dfrac{N_k}{N}$
  $\mu_k = \dfrac{1}{N_k} \sum_{n=1}^{N} r_{nk} x_n$
  $\Sigma_k = \dfrac{1}{N_k} \sum_{n=1}^{N} r_{nk} (x_n - \mu_k)(x_n - \mu_k)^T$
until the parameter updates satisfy $\|\theta^{(t)} - \theta^{(t-1)}\| < \epsilon$

2.1.2. Bayesian Clustering

In a standard GMM, we compute only point estimates for the model parameters, such as the means μ_k, covariances Σ_k, and mixing weights π_k. However, these estimates do not capture uncertainty. In contrast, Bayesian GMM models uncertainty by placing prior distributions over the parameters, treating them as random variables. The Bayesian plate diagram 2.1b reflects this by including additional nodes and edges for these priors.

The prior over the mixing proportions is a Dirichlet distribution:

$$p(\pi) = \mathsf{Dir}(\pi|\alpha_0) = C(\alpha_0) \prod_{k=1}^{K} \pi_k^{\alpha_0 - 1} \tag{2.8}$$

where $C(\alpha_0)$ is a normalization constant that ensures the mixing proportions sum up to 1.

For each component k, the prior over the mean μ_k and inverse covariance matrix Σ_k^{-1} is a Normal-Wishart distribution:

$$p(\mu, \Sigma) = \prod_{k=1}^{K} \mathcal{N}(\mu_k|m_0, \beta_0^{-1}\Sigma_k)\mathcal{W}(\Sigma_k^{-1}|W_0, \nu_0) \tag{2.9}$$

These priors are conjugate to the Gaussian likelihood, which ensures that the posterior distri-

bution remains in the same family, simplifying inference [9]. For a more in-depth formulation of these distributions, refer to Appendix A.

The full joint distribution of the Bayesian GMM can be derived from figure 2.1b as:

$$p(x, z, \pi, \mu, \Sigma) = p(x|z, \mu, \Sigma)p(z|\pi)p(\pi)p(\mu|\Sigma)p(\Sigma) \tag{2.10}$$

In contrast to the standard GMM where we can compute the posterior cluster assignments $p(z|x, \pi, \mu, \Sigma)$ directly, Bayesian inference requires computing the full posterior:

$$p(\pi, \mu, \Sigma, z \mid x) \tag{2.11}$$

This involves integrating over all latent variables and parameters, which becomes computationally infeasible due to the high dimensionality and complex dependencies between variables. Thus, exact inference is intractable.

This introduces us to VI as we have to use it to approximate this posterior. The idea is to introduce a simpler, tractable distribution $q(\pi, \mu, \Sigma, z)$, and optimize it to be as close as possible to the true posterior.

To make this optimization feasible, we make an assumption on the structure of $q(z)$ such that it factorizes across variables. It is known as the mean-field approximation and is written as:

$$q(\boldsymbol{z}) = \prod_i q_i(z_i) \tag{2.12}$$

Each index i here denotes a subset or group of latent variables. This assumption simplifies dependencies, allowing each factor to be updated iteratively and independently. In the Bayesian GMM case, we consider this factorization:

$$q(z, \pi, \mu, \Lambda) = q(z)q(\pi, \mu, \Sigma) \tag{2.13}$$

The result is an algorithm similar to EM where rather than updating point estimates of parameters, VI alternates between updating the variational distributions over latent variables and parameters until convergence. However, because the mean-field approximation imposes restrictions on the form of these distributions, the variational distribution does not necessarily converge to the true posterior. Instead, it converges to the best possible approximation within the constraints of the variational family.

We refer to this Bayesian clustering algorithm as Bayesian GMM with VI (Bayesian GMM VI). For more details on Bayesian GMMs, refer to Bishop's textbook [9].

2.2. Estimating Number of Clusters

Several heuristic methods are commonly used to estimate the number of clusters in a dataset. In this work, we focus on the Akaike Information Criterion (AIC), the Bayesian Information

Criterion (BIC), and the silhouette score, as these are widely used for clustering comparisons [1, 26].

Both AIC and BIC are model selection criteria where lower values indicate better models by balancing goodness of fit with model complexity.

The AIC is computed as:

$$\text{AIC} = 2p - 2\ln(L)$$

where p is the number of estimated parameters and L is the maximized value of the likelihood function (Equation 2.4). Similarly, BIC is calculated by:

$$\text{BIC} = p\ln(N) - 2\ln(L)$$

where N is the total number of data points. Compared to AIC, BIC imposes a stronger penalty on model complexity, especially for larger datasets.

To define the silhouette score, we first introduce a few variables. For each data point i in cluster C_I, let

$$a(i) = \frac{1}{|C_I| - 1} \sum_{\substack{j \in C_I \\ j \neq i}} d(i, j)$$

where $a(i)$ measures the average distance between point i and all other points in its own cluster. A smaller value of $a(i)$ indicates that i is well-matched to its cluster.

Next, we define

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i, j)$$

which represents the minimum average distance from point i to points in any other cluster C_J. This captures the dissimilarity between point i and its nearest neighboring cluster.

The silhouette score for point i is then given by:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

if $|C_I| > 1$, and $s(i) = 0$ otherwise.

The final silhouette score is the average of the individual silhouette scores for all data points. Higher silhouette scores are preferred, as they indicate that data points are well clustered.

To estimate the number of clusters using these heuristics, we fit a GMM to the dataset over a range of cluster values and select the number of clusters that yield the best score according to each heuristic.

The Bayesian GMM VI provides probabilistic assignments of data points to clusters. To estimate the number of clusters, we assign each point to the cluster with the highest probability and then count the total number of unique clusters represented across all points.

2.3. Transformers and In-Context Learning

The transformer [39] is a deep learning model designed for sequence modeling tasks. Its power lies in the attention mechanism, which enables effective information sharing across sequences. This allows the model to capture complex relationships between data.

Interesting behavior observed by Brown et al. [10] have shown that deep learning models, when trained with sufficient data and scale [12], can perform in-context learning (ICL). In this setting, a model that does not undergo any additional parameter updates can learn to make accurate predictions purely by being shown a sequence of input-output examples during inference time.

More specifically, in ICL, the model is first pre-trained on a diverse set of problems. Through this training, it learns to recognize and adapt to the structure of the problem, a process known as meta-learning. At test time, the model is presented with a sequence of input-output pairs $(x_i, f(x_i))_{i=1}^n$, followed by a new query input x'. Based on this context, the model is able to produce the correct prediction $f(x')$. This new prediction occurs without any updates to the model's parameters, in contrast to traditional supervised learning where learning requires explicit parameter optimization through gradient descent.

The transformer architecture has shown to be particularly effective at performing ICL, and it has been shown to learn algorithms such as linear and logistic regression [3, 21]. This suggests that it may be capable of learning a wider range of algorithms.

2.4. Prior-Data Fitted Networks

Müller et al. [30] have combined ICL with transformers to develop Prior-Data Fitted Networks (PFNs), teaching transformers how to do Bayesian inference. In particular, the PFN model can directly approximate the PPD (equation 2.2) in one forward pass.

2.4.1. Problem Setup

The procedure is as follows: a prior over problems is defined as a sampling distribution. Each problem specifies a relationship between the input and output. Once a problem is sampled, a dataset is sampled from the problem and formatted as $D_{\text{train}}, x_{\text{test}}$, where D_{train} is the training data, consisting of features X_{train} and outputs Y_{train}. x_{test} is an unseen input, and y_{test} is the corresponding label. The PFN is then given $D_{\text{train}}, x_{\text{test}}$ and predicts the distribution for x_{test}.

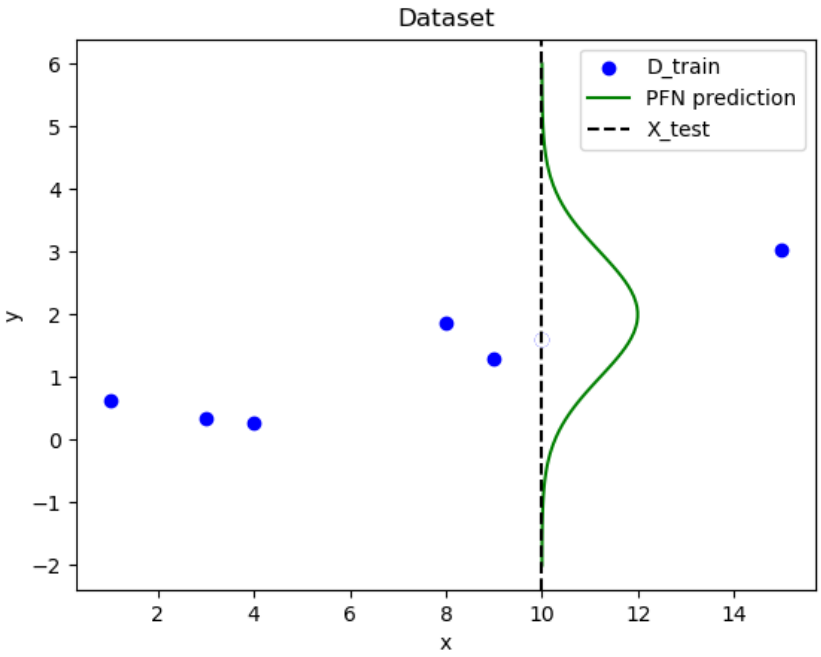


Figure 2.2: Example of a sampled problem

Figure 2.2 illustrates an example of a sampled problem. The prior is defined as a sampling distribution that generates noisy linear regression curves. Therefore, each problem corresponds to a specific realization of such a regression.

The training dataset D_{train} is visualized as blue data points, and x_{test} is indicated by the vertical dashed line. The PFN takes D_{train} and x_{test} as input and predicts a distribution over possible values for y_{test}, visualized by the green distribution. The prediction is then compared to the true value y_{test}, and the model is trained to minimize a loss similar to the negative log-likelihood (NLL) of the true label under the predicted distribution.

Formally, the PFN is trained to minimize the following loss function:

$$\ell_\theta = \mathbb{E}_{\mathcal{D} \cup \{x_{test}, y\} \sim p(\mathcal{D})} \left[-\log q_\theta(y \mid x_{test}, \mathcal{D}) \right] \tag{2.14}$$

where q_θ denotes the model parameterized by θ.

Müller et al. [30] formalize that the PFN indeed approximates the PPD by proving that the loss objective ℓ_θ is equal to the expectation of the cross-entropy between the PPD and its approximation:

$$\ell_\theta = \mathbb{E}_{x_{test}, D \sim p(D)} \left[\mathcal{H} \left(p(\cdot \mid x_{test}, D),\ q_\theta(\cdot \mid x_{test}, D) \right) \right]$$

Here, $\mathcal{H}$ denotes the cross-entropy between the true PPD $p(\cdot \mid x_{test}, D)$ and the learned approximation $q_\theta(\cdot \mid x_{test}, D)$.

2.4.2. PFN Architecture and Training Procedure

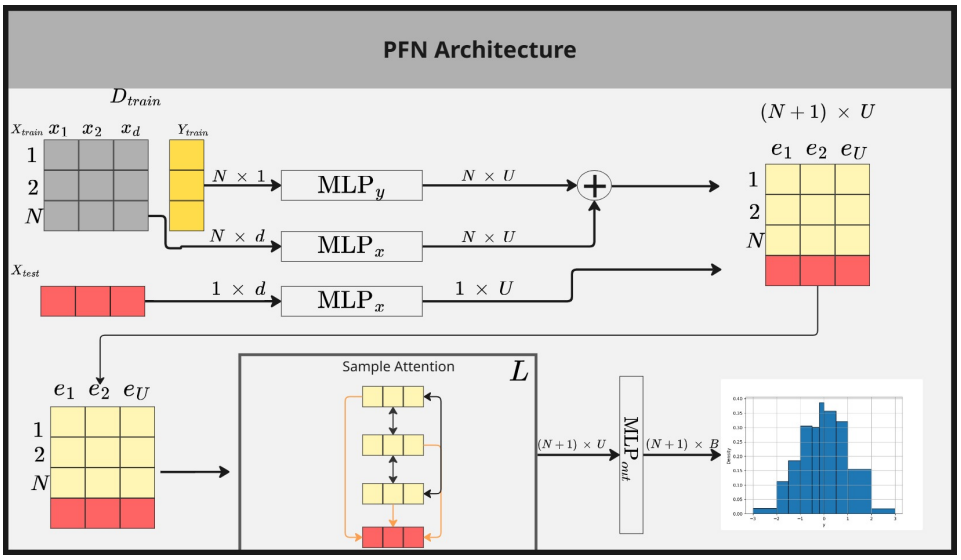


Figure 2.3: Full training pipeline of the PFN

Figure 2.3 illustrates the PFN architecture. The input consists of a training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ are the feature vectors and $y_i \in \mathbb{R}$ are the corresponding targets. The training dataset is split into feature inputs $X_{\text{train}} = \{x_i\}_{i=1}^{N}$ and labels $Y_{\text{train}} = \{y_i\}_{i=1}^{N}$. Along with this, the model is provided with a query input x_{test}, for which it must predict a distribution over y_{test}.

Step 1: Input Embedding. The training inputs $X_{\text{train}} \in \mathbb{R}^{N \times d}$ and test input $x_{\text{test}} \in \mathbb{R}^d$ are passed through a shared MLP:

$$\mathbf{E}^x_{\text{train}} = \text{MLP}_x(X_{\text{train}}) \in \mathbb{R}^{N \times U}, \quad \mathbf{e}^x_{\text{test}} = \text{MLP}_x(x_{\text{test}}) \in \mathbb{R}^{1 \times U}$$

The training targets $Y_{\text{train}} \in \mathbb{R}^{N \times 1}$ are embedded separately:

$$\mathbf{E}^y_{\text{train}} = \text{MLP}_y(Y_{\text{train}}) \in \mathbb{R}^{N \times U}$$

Step 2: Combining Inputs. The embedded training features and targets are combined element-wise:

$$\mathbf{E}_{\text{train}} = \mathbf{E}^x_{\text{train}} + \mathbf{E}^y_{\text{train}} \in \mathbb{R}^{N \times U}$$

Each vector $\mathbf{e}_i \in \mathbf{E}_{\text{train}}$ represents the joint embedding of the i-th input-output pair. The embedded test input is appended to form the full input sequence:

$$\mathbf{E} = \begin{bmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_N \\ \mathbf{e}^x_{\text{test}} \end{bmatrix} \in \mathbb{R}^{(N+1) \times U}$$

Step 3: Transformer Encoder. The combined sequence $\mathbf{E} \in \mathbb{R}^{(N+1)\times U}$ is passed through a Transformer encoder. Self-attention is applied across the input embeddings, using a custom mask m that enforces the following constraints:

- Each training point can attend to all other training points.
- The test point can attend to all training points and to itself.
- Training points cannot attend to the test point.

The output of the Transformer encoder is denoted by

$$\mathbf{H}^{\text{enc}} = \text{Encoder}(\mathbf{E}, m) \in \mathbb{R}^{(N+1)\times U}.$$

Step 4: Prediction. The encoder outputs are passed through a final MLP to produce logits for a discretized distribution over target values. Each embedding is mapped to B output bins:

$$\mathbf{Y} = \text{MLP}_{\text{out}}(\mathbf{H}^{\text{enc}}) \in \mathbb{R}^{(N+1)\times B}.$$

Only the final row $\mathbf{Y}_{N+1} \in \mathbb{R}^B$, corresponding to the test point, is used for prediction. This vector represents a categorical distribution over the possible values of y_{test}, where each bin corresponds to a discretized range of values. Thus, the prediction task is framed as a classification problem over value intervals for the target.

2.4.3. PFN Strengths and Limitations

The PFN offers several strengths. First, the model is simple, computationally efficient, and easily adaptable to a wide range of priors. Second, it can approximate the PPD of certain functions significantly faster than traditional methods such as MCMC and VI.

One of the limitations of the PFN model is its fixed maximum input feature size, which restricts flexibility for handling arbitrary-dimensional data. Additionally, the algorithm scales quadratically with the number of samples, making it computationally expensive for large datasets [32].

3

Cluster-PFN

PFNs have demonstrated the capability to perform Bayesian inference when trained on a prior. A natural extension of this idea is to explore whether a PFN can also learn to perform Bayesian clustering.

3.1. Objectives

We hypothesize that the Cluster-PFN can perform Bayesian clustering. Specifically, given a dataset $X = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ and corresponding latent variables $y_i \in \{0, \ldots, K-1\}$, where K (the number of clusters) is unknown, the objective is to estimate the posterior distribution over cluster assignments $p(y_i \mid X)$ for each data point.

We adapt the loss function presented in [30], where the training objective minimizes the NLL of the target variable $Y = \{y_1, \ldots, y_n\}$, representing the cluster assignments, conditioned on the input data X:

$$\ell_\theta = \mathbb{E}_{X \sim P(D)} \left[-\frac{1}{n} \sum_{i=1}^{n} \log q_\theta(y_i \mid X) \right] \tag{3.1}$$

where $q_\theta(y_i \mid X)$ is the model's predicted distribution over cluster labels. This loss penalizes the model for assigning low probability to the true clustering labels. With a minor adaptation from the formulation in [30], this objective is equivalent to minimizing the expected cross-entropy between the true posterior distribution $p(\cdot \mid X)$ and the model's approximation $q_\theta(\cdot \mid X)$:

$$\ell_\theta = \mathbb{E}_{X \sim P(D)} \left[\frac{1}{n} \sum_{i=1}^{n} \mathcal{H}\left(p(\cdot \mid X),\, q_\theta(\cdot \mid X)\right) \right] \tag{3.2}$$

Thus, the Cluster-PFN learns to approximate the posterior over cluster assignments given the

observed data.

In addition to cluster assignments, we also aim to approximate the posterior distribution over the number of clusters, $p(K \mid X)$, given a dataset X. Using an analogous loss formulation, we hypothesize that the Cluster-PFN can learn to predict the number of clusters by minimizing the expected cross-entropy between the true and predicted distributions over cluster counts.

Finally, we aim to enable the Cluster-PFN to generate cluster assignments conditioned on a user-specified number of clusters, allowing users to obtain clusterings that reflect their preferences during inference.

3.2. Architecture

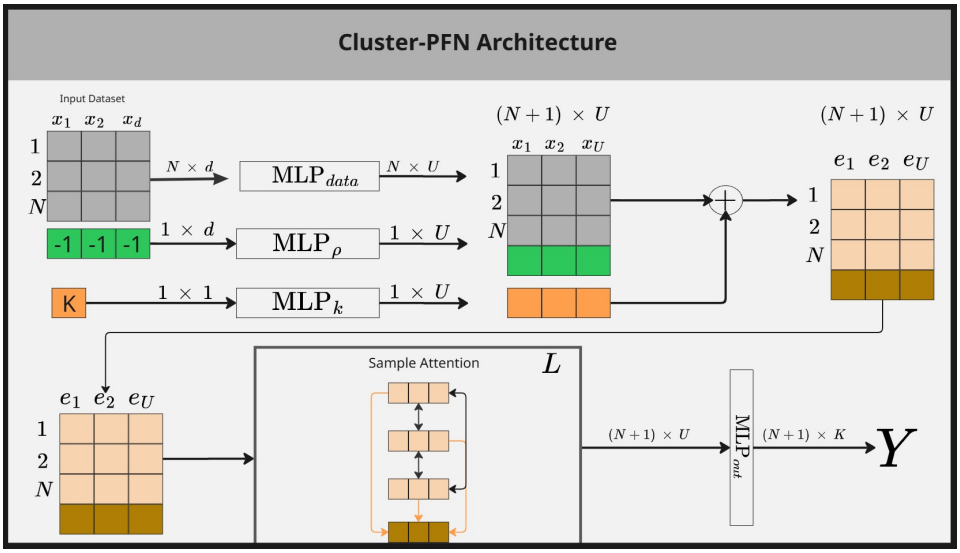


Figure 3.1: Full training pipeline of the Cluster-PFN

The full training pipeline is illustrated in figure 3.1. As shown, we begin with the input data $X \in \mathbb{R}^{N \times d}$ where N is the number of samples and d is the number of features. We also include an additional input variable K, which denotes the number of clusters in the dataset X. During training, K is set as follows:

$$K = \begin{cases} \text{true number of clusters} & \text{with probability } 0.5, \\ 0 & \text{with probability } 0.5, \end{cases}$$

This design allows the model to operate in both supervised and unsupervised settings. Specifically, setting $K = 0$ encourages the transformer to infer the number of clusters on its own, while providing the true value of K enables it to generate cluster assignments conditioned on that number. In this way, K serves as a conditioning mechanism, allowing users to specify the desired number of clusters during inference.

Internally in the model, A vector of -1s of dimension d is defined, denoted as $\rho = -\mathbf{1} \in \mathbb{R}^d$

which serves as a special learnable token used to predict the number of clusters. The vector is filled with -1 because, after normalization, these are the only negative values in the input space, making ρ easily distinguishable from the actual data points.

Step 1: Input Embedding. We begin by projecting the dataset into a higher-dimensional space using an MLP:

$$\mathbf{E}^x = \mathsf{MLP}_{\mathsf{data}}(X) \in \mathbb{R}^{N \times U}.$$

The number of clusters K is passed through an MLP:

$$\mathbf{e}_K = \mathsf{MLP}_k(K) \in \mathbb{R}^{1 \times U}.$$

The internally defined vector ρ, is also passed through an MLP to obtain a learnable embedding:

$$\tilde{\rho} = \mathsf{MLP}_\rho(\rho) \in \mathbb{R}^{1 \times U}.$$

Step 2: Input Combination. We append $\tilde{\rho}$ to the end of $\mathbf{E}^x$ resulting in:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E}^x \\ \tilde{\rho} \end{bmatrix} \in \mathbb{R}^{(N+1) \times U}.$$

Next we perform a row-wise addition of e_k to each sample in $\mathbf{E}$

$$\mathbf{E}'_{i,j} = \mathbf{E}_{i,j} + e_{k,j} \quad \forall i \in \{1, \ldots, N\}, \quad j \in \{1, \ldots, U\}$$

Step 3: Transformer Encoder. The input $\mathbf{E}'$ is passed through a transformer encoder. Self-attention is applied across the input embeddings, using a custom attention mask, m, that enforces the following constraints:

- Each data point can attend to all other data points.
- The final input (i.e., ρ) can attend to all data points including itself.
- Data points cannot attend to ρ.

The output of the encoder is denoted by:

$$\mathbf{H}^{(\mathsf{enc})} = \mathsf{Encoder}(\mathbf{E}', m) \in \mathbb{R}^{(N+1) \times U}.$$

This design allows $\tilde{\rho}$ to aggregate global information for predicting the number of clusters while keeping individual cluster assignments unaffected by $\tilde{\rho}$.

Step 4: Prediction. The encoder outputs are passed through a final MLP that maps each sample to K outputs:

$$\mathbf{Y} = \mathsf{MLP}_{\mathsf{out}}(\mathbf{H}^{(\mathsf{enc})}) \in \mathbb{R}^{(N+1) \times K}.$$

For data points $i = 1, \ldots, N$, each $\mathbf{Y}_i$ represents a distribution over cluster memberships. The final row $\mathbf{Y}_{N+1}$ corresponds to a distribution over possible cluster counts. The last output is only meaningful when $K = 0$; otherwise, the user-specified value of K is used for conditioning.

3.3. Research Questions

We now outline the core research questions that guide our investigation.

- **RQ1: Can the Cluster-PFN cluster synthetic and real-world data?** This question evaluates the model's ability to cluster both synthetic and real-world datasets, as well as its capability to generate cluster assignments conditioned on a user-specified number of clusters.
- **RQ2: How does the Cluster-PFN compare to GMM (with AIC, BIC, or silhouette-based heuristics) and Bayesian GMM VI in estimating the number of clusters?** This question investigates how well the Cluster-PFN infers the cluster counts compared to existing heuristic methods to estimate the number of clusters in a dataset.
- **RQ3: How does the Cluster-PFN perform against Bayesian GMM VI in terms of clustering quality, uncertainty estimation, and inference speed?** Given that the Cluster-PFN produces probabilistic outputs of cluster assignments, we evaluate its performance against existing Bayesian clustering methods.
- **RQ4: How does the Cluster-PFN perform against traditional GMM and K-means++ in clustering quality?** We evaluate how the Cluster-PFN compares against the GMM and KMeans++ in clustering quality to assess the Cluster-PFN's general performance.

4

Experimental Setup

This chapter outlines the procedure for the dataset generation as well as the metrics used to evaluate the performance of the Cluster-PFN.

4.1. Dataset Generation

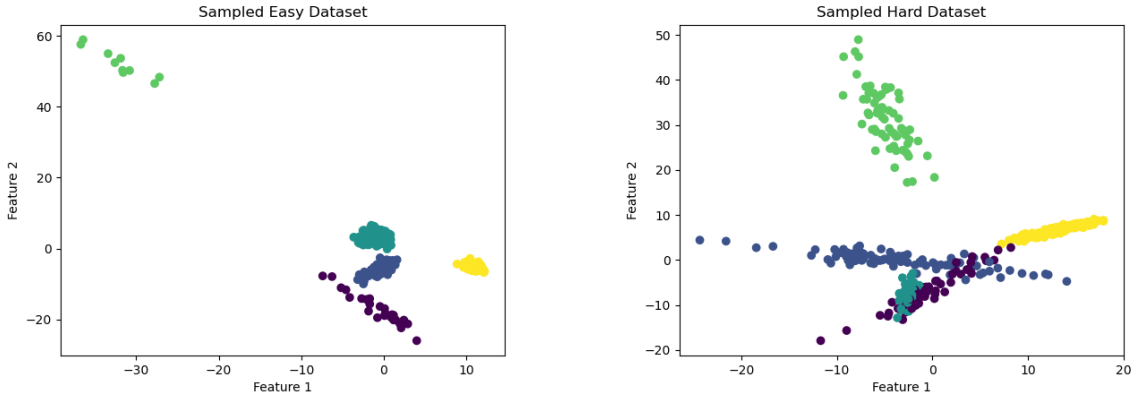
The data generation process follows the approach illustrated in Figure 2.1b. To construct the dataset, we first uniformly sample the number of data points from a defined range. Then, we uniformly sample a number k from 1 to some upper bound K representing the number of clusters we want to generate. Next, we assign mixture weights π_k, to each cluster, ensuring that the weights add up to one. These weights determine the proportion of data points that should belong to each cluster. As described in Section 2.1.2, the mixture weights are drawn from a Dirichlet distribution.

For each cluster, we sample a covariance matrix and a mean vector, which are then used to generate data points. The means and covariances are sampled from a Gaussian-Wishart distribution. The hyperparameters m_0, β_0, W_0, and v_0 are set to fixed values, with β_0 varied across experiments.

Once the data is generated, in order to feed it to the Cluster-PFN, we normalize the feature values to lie within the range of 0 and 1. An important thing to note is that labels assigned in cluster datasets are arbitrary. For example, a generated dataset with three clusters can have labels {0, 1, 2} while another equivalent clustering can have labels {2, 0, 1}. Although these labelings represent the same clustering, the Cluster-PFN treats them as entirely different, which can prevent it from learning the correct structure. To address this, we apply a deterministic relabeling scheme to enforce a consistent label ordering. Specifically, we identify the point closest to the origin and assign its cluster the label 0. All data points in that cluster are then relabeled accordingly. We repeat this process by finding the next closest unassigned point and assigning its cluster the next available label (e.g., label 1), continuing until all clusters are relabeled. This preprocessing step ensures a consistent labeling scheme allowing the

Cluster-PFN to learn more effectively.

We train four distinct Cluster-PFN models. Two models are trained on two-dimensional feature inputs (2D), while the other two are trained on inputs ranging from two to five dimensions (5D). For the 5D case, we generate datasets with dimensions from two to five and pad the remaining dimensions with zeros so that all inputs are represented in five-dimensional space before being fed into the Cluster-PFN.



(a) Sampled Easy problem

(b) Sampled Hard problem

Figure 4.1: Examples of Easy and Hard problem settings

Within each feature group, we train the models using two different hyperparameter settings. Specifically, we vary the β parameter, using values of 0.01 and 0.1, which result in clusters that are either more spread out or more interspersed, respectively. In Section 5, we refer to these settings as Easy and Hard problems: Easy corresponds to Cluster-PFN trained on more spread-out clusters, while Hard refers to Cluster-PFN trained on more interspersed clusters. An example of these problem types is shown in Figure 4.1. For the remaining hyperparameters, we fix $\alpha = 1$, set the mean $m = 0$, use the identity matrix for W, and set v equal to the number of features. The maximum number of clusters generated, K, is set to 10. To evaluate the performance of these models, we sample unseen datasets from the respective priors (following the same distributions) and compare their performance against other clustering algorithms.

Models trained on two-dimensional features are trained for 600,000 epochs (60 million synthetic datasets), requiring approximately 40 GPU hours. Models trained on two- to five-dimensional features are trained for 900,000 epochs (90 million datasets), taking around 60 GPU hours. All experiments were conducted on the Delft AI Cluster, utilizing a single NVIDIA L40 GPU.

For more details about the architectural setup and training, please refer to Appendix B.

4.2. External Metrics

External evaluation refers to clustering results that are evaluated when given the true cluster labels. In our case, we have the true cluster labels due to the synthetic data generation. One of the external metrics we use to evaluate clustering performance is the Adjusted Rand Index (ARI). To understand ARI, we first look at the Rand Index (RI), defined as:

$$\text{RI} = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

The Rand Index measures the similarity between clusterings by examining pairs of data points:

- **TP (True Positives)**: Pairs of points that are in the same cluster in both the predicted and true clusterings.
- **TN (True Negatives)**: Pairs of points that are in different clusters in both the predicted and true clusterings.
- **FP (False Positives)**: Pairs of points that are in the same cluster in the prediction, but in different clusters in the ground truth.
- **FN (False Negatives)**: Pairs of points that are in different clusters in the prediction, but in the same cluster in the ground truth.

Intuitively, the Rand Index evaluates the proportion of decisions (about whether a pair of points should be in the same cluster or not) that the clustering algorithm got correct. However, since the RI does not account for chance groupings, the Adjusted Rand Index introduces a normalization that adjusts the score for random labelings [25].

Another external metric we use is the Adjusted Mutual Information (AMI). It is a measure of the similarity between two labels of the same data. The formula is given by:

$$\text{MI}(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \left(\frac{N|U_i \cap V_j|}{|U_i||V_j|} \right) \tag{4.2}$$

Where:

- $U = \{U_1, U_2, \ldots, U_{|U|}\}$ is the set of ground truth clusters.
- $V = \{V_1, V_2, \ldots, V_{|V|}\}$ is the set of predicted clusters.
- N is the total number of data points.
- $|U_i \cap V_j|$ is the number of data points that appear in both cluster U_i and cluster V_j.
- $|U_i|$ and $|V_j|$ are the sizes of clusters U_i and V_j, respectively.

The higher the MI, the better the clustering. However, MI is not normalized and can be biased by the number of clusters. AMI corrects this bias by subtracting the expected MI of random clusterings and normalizing the result [37].

Although these metrics have similar behavior. They are not exactly the same. ARI excels when clusters are large equal-sized clusters and AMI is best when clustering is unbalanced and there are small clusters [38]. Our data generation yields both of these types of clusters and therefore they are both valid metrics to use.

The third external metric we use is purity. It is defined as:

$$\text{Purity} = \frac{1}{N} \sum_{k=1}^{K} \max_{j} |C_k \cap L_j| \tag{4.3}$$

where N is the total number of data points and K is the number of clusters. C_k is the set of data points in cluster k, and L_j as the set of data points with ground truth label j. The term $|C_k \cap L_j|$ represents the number of data points in cluster k that belong to ground truth class j. In essence, purity measures the extent to which clusters contain data points predominantly from a single true class. A high purity value indicates that most points within each cluster belong to the same class, reflecting better clustering performance.

Although these three metrics evaluate the final assignment performance of the data point, in order to evaluate the probabilistic aspect, we need to compare the probabilities the cluster-PFN gives for each data point against the Bayesian GMM VI. In order to do that we will be using the negative log-likelihood loss (NLL) as it measures the probabilistic performance accurately. The formula for NLL is:

$$l_{NLL} = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(p_{ik}) \tag{4.4}$$

We sum over all data points and clusters. For each input, y_{ik} is a binary indicator that is 1 if the sample, i belongs to the cluster k and zero otherwise, and p_{ik} is the predicted probability that sample i belongs to cluster k. Since dataset sizes can vary, we use the average NLL as our loss metric. As our true labels will not necessarily match with the labels from the models, we will permute the labels and retrieve the lowest NLL given by both of the models.

To compare the performance of different models on external metrics, we sample a set of datasets and analyze the distribution of their scores across various metrics. Additionally, we compute pairwise wins of the models by counting how often each model outperforms the other on each external metric across the sampled datasets. We refer to this in the results as win rates.

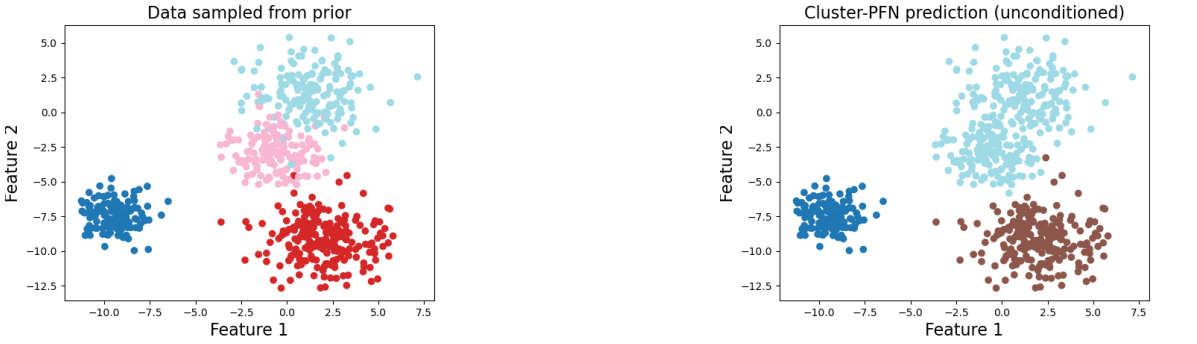
5

Results

5.1. RQ1: Can the Cluster-PFN cluster synthetic and real-world data?

5.1.1. Cluster-PFN on Synthetic Data

We qualitatively analyze the clustering assignments produced by the Cluster-PFN on both synthetic and real-world datasets. As the Cluster-PFN outputs a probability distribution over possible cluster assignments, each data point is assigned to the cluster with the highest probability, resulting in hard cluster labels.



(a) Dataset sampled from prior

(b) Cluster-PFN cluster assignment unconditioned

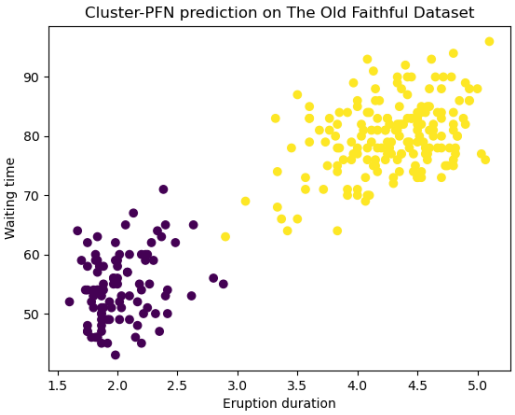
Figure 5.1: Synthetic dataset sampled from 2D Easy prior with Cluster-PFN prediction

Figure 5.1a shows an example of an unseen synthetic dataset, while the corresponding prediction by Cluster-PFN without conditioning is shown in Figure 5.1b. In this case, the original dataset contains four clusters, but Cluster-PFN contains 3, however, the Cluster-PFN still produces a reasonable clustering.

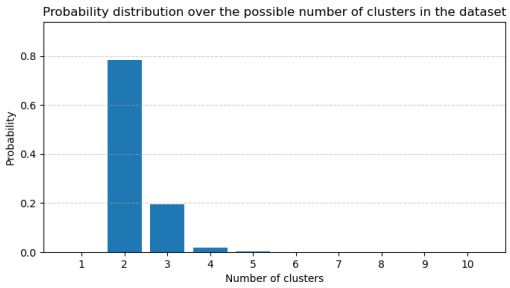
5.1.2. Cluster-PFN on Real Data

We benchmark the Cluster-PFN on two real-world datasets: The Old Faithful and Iris datasets. The Old Faithful is an unlabeled geyser dataset that records eruption duration and waiting

time [13]. The Iris dataset contains 150 labeled samples of iris flowers with sepal and petal measurements, commonly used for evaluating clustering and classification methods [17, 19].



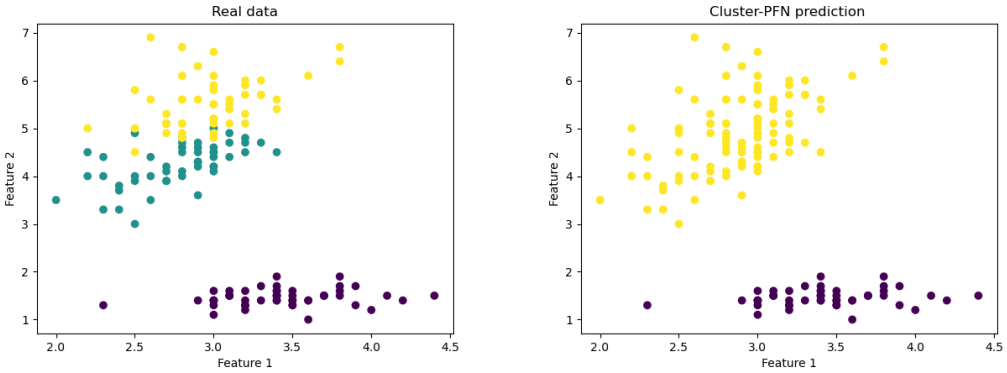
(a) Cluster-PFN cluster assignment prediction



(b) Cluster-PFN number probability distribution over the number of possible clusters

Figure 5.2: Cluster-PFN prediction for the Old Faithful Dataset (Cluster-PFN trained on Easy setting)

Figure 5.2 displays the Cluster-PFN's prediction for the Old Faithful dataset. The model identifies two distinct clusters and assigns a high probability to the dataset having two clusters. Overall, Cluster-PFN gives a reasonable clustering to the data.



(a) Iris dataset with labels

(b) Cluster-PFN prediction on the Iris dataset

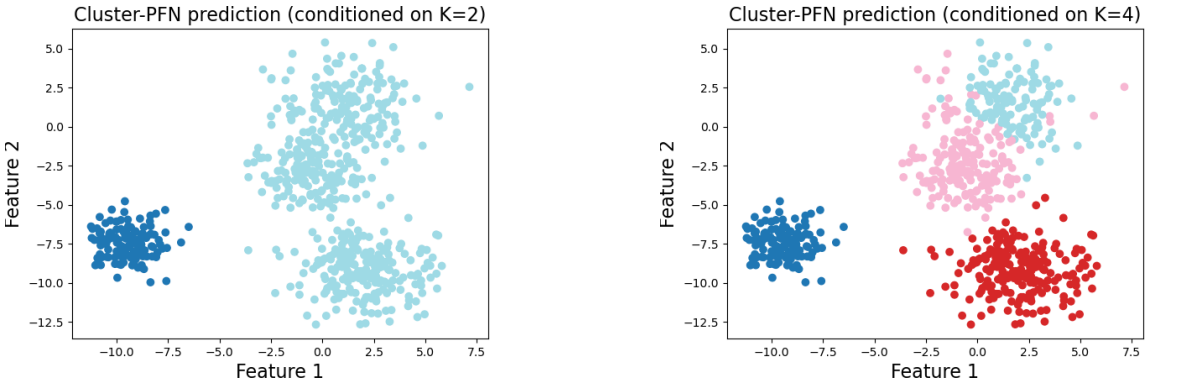
Figure 5.3: Cluster-PFN results on the Iris dataset. (a) shows the true class labels, while (b) shows the model's prediction (Cluster-PFN trained on hard setting)

Figure 5.3 shows the Iris dataset alongside the Cluster-PFN predictions. Since the dataset contains four features, we visualize only two for clarity in Figure 5.3a. Figure 5.3b presents the cluster assignments predicted by the Cluster-PFN. Although the Cluster-PFN does not perfectly capture all the underlying class distinctions, it still produces a reasonable and coherent clustering structure.

Since the Iris dataset has ground truth labels, we can evaluate the performance of the Cluster-PFN using external metrics. The Cluster-PFN achieves an AMI score of 0.73, an ARI score of 0.77, and a purity score of 1.

5.1.3. Cluster-PFN on Conditioning

We can also qualitatively evaluate the model's behavior when conditioned on a specified number of clusters. Using the same data from Figure 5.1 when the model is conditioned to produce two clusters, it outputs two distinct clusters, as shown in Figure 5.4a. Similarly, when conditioned to produce four clusters, it correctly identifies four distinct clusters, as seen in Figure 5.4b.

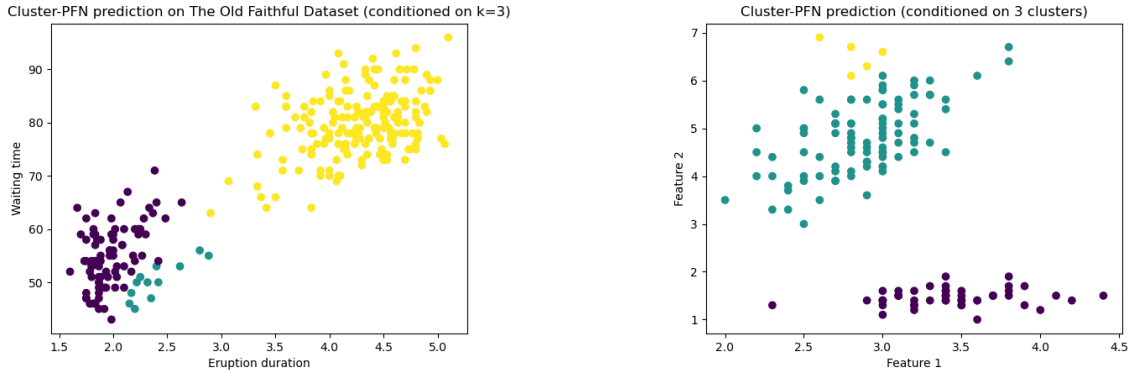


(a) Cluster-PFN cluster assignment conditioned on 2 clusters

(b) Cluster-PFN cluster assignment conditioned on 4 clusters

Figure 5.4: Cluster-PFN conditioned predictions on different cluster amounts

Furthermore, we can also apply the conditioning to our real-world datasets.



(a) Cluster-PFN cluster assignment on Old Faithful conditioned on 3 Clusters

(b) Cluster-PFN cluster assignment on Iris conditioned on 3 clusters

Figure 5.5: Cluster-PFN conditioned predictions on different cluster amounts for Old Faithful and Iris datasets

Figure 5.5a shows the Cluster-PFN's predictions on the Old Faithful dataset when conditioned on 3 clusters, while Figure 5.5b presents the Cluster-PFN's predictions on the Iris dataset under the same condition.

These results demonstrate that the Cluster-PFN can adapt its clustering behavior based on the specified number of clusters. However, the model does not always strictly adhere to the provided cluster count.

To quantitatively evaluate the conditioning accuracy, we sampled 30,000 synthetic datasets and provided a randomly selected cluster count as the conditioning input to the Cluster-PFN. We then measured how closely the predicted number of clusters matched the specified condition. This was done by first obtaining the hard labels from the Cluster-PFN's predictions and then counting the number of unique labels. The evaluation was performed under varying thresholds:

- A threshold of 0 means the model predicted exactly the conditioned number of clusters.
- A threshold of 1 allows for a deviation of ±1 from the condition.
- A threshold of 2 allows for a deviation of ±2 from the condition.

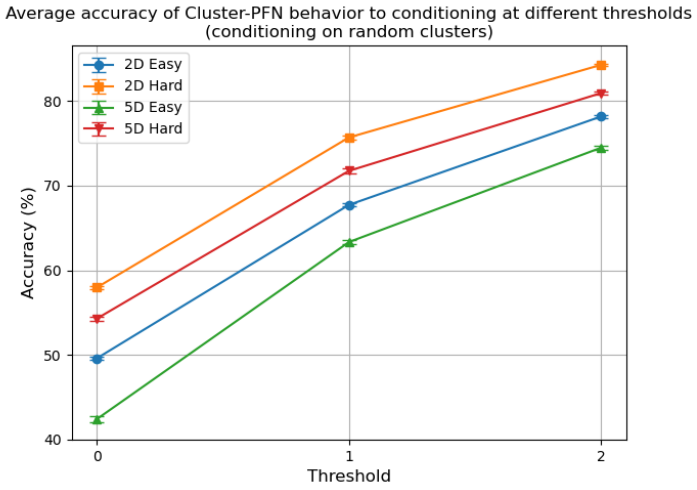


Figure 5.6: Cluster-PFN listening to condition at different thresholds (conditioning on random clusters)

Figure 5.6 shows the accuracy across different thresholds. In the Hard setting, the Cluster-PFN adheres to the conditioning slightly better than in the Easy setting. However, the conditioning is generally ineffective when using randomly specified cluster numbers, as evidenced by the highest accuracy at threshold 0 being just under 60%.

However, randomly sampling condition cluster values may not be the most meaningful way to evaluate the model's conditioning ability, as some randomly assigned conditions may differ substantially from the Cluster-PFN's natural prediction. In such cases, forcing the model to deviate significantly from its prediction is not very sensible.

To address this, we conduct a second experiment. Instead of randomly selecting conditions, we begin with the model's unconditioned prediction and perturb it by ±1 cluster. We then evaluate how well the Cluster-PFN adapts to these nearby, more reasonable conditioning values.

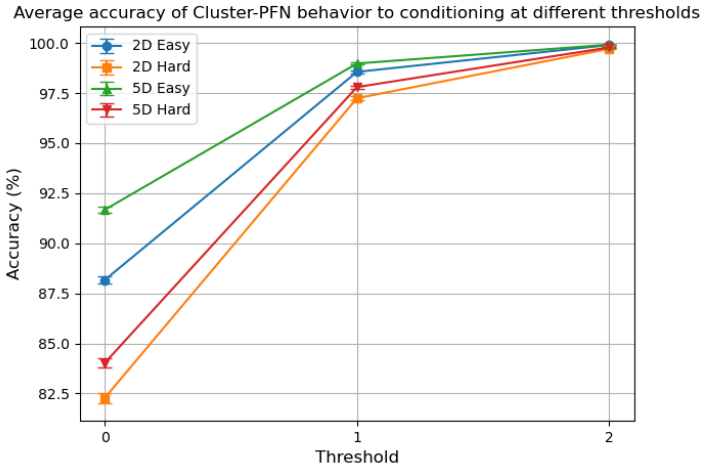


Figure 5.7: Cluster-PFN listening to condition at different thresholds (conditioning on more appropriate clusters)

We observe significantly higher accuracy when the conditioning values are more reasonable. The model already achieves strong accuracy at a threshold of 0, with a sharp increase in performance at a threshold of 1, and reaches perfect accuracy at the final threshold.

5.2. RQ2: Can the Cluster-PFN predict a suitable number of clusters?

Since the Cluster-PFN can also predict the number of clusters, we sample 30,000 datasets and compare its predictions with those from the Bayesian GMM VI and the traditional GMM. Because the standard GMM requires the number of clusters to be specified, we iterate over a range of cluster counts from 2 to 10 (inclusive) and compute the AIC, BIC, and silhouette scores to determine the optimal number of clusters.

Model	2D Easy	5D Easy	2D Hard	5D Hard
Cluster-PFN	**64**	**71**	**43**	**51**
GMM (AIC)	41	46	30	33
GMM (BIC)	39	43	28	27
GMM (Silhouette)	41	46	29	33
Bayesian GMM VI	18	35	18	25

Table 5.1: Percentage of correct clusters predicted by different models across 30,000 datasets in various experiments.

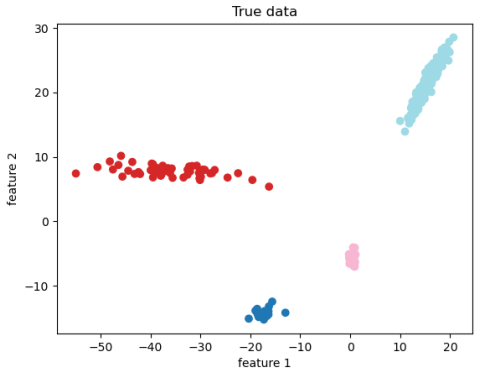
Table 5.1 presents the percentage of correctly predicted clusters by the different models. Cluster-PFN consistently outperforms all baseline methods, achieving the highest accuracy in every scenario. Traditional GMM methods (using AIC, BIC, and Silhouette criteria) perform moderately well but lag behind Cluster-PFN. The Bayesian GMM VI shows the weakest performance overall, particularly in the 2D cases. These results highlight the superior ability of Cluster-PFN to accurately recover the true number of clusters across varying levels of difficulty and dimensionality.

5.3. RQ3: How does the Cluster-PFN perform against Bayesian GMM VI?

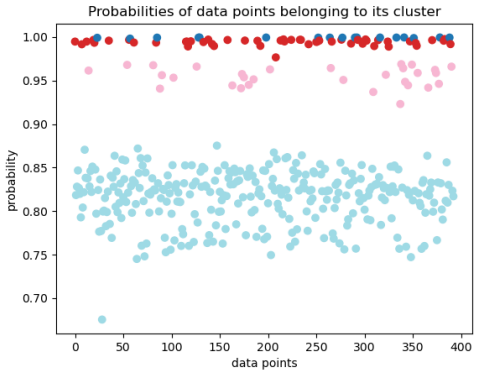
5.3.1. Overview and Conditioning in Cluster-PFN

To compare the performance of the Cluster-PFN with the Bayesian GMM VI, we use external evaluation metrics to assess the clustering quality of both models. However, it is important to note a key characteristic of the Cluster-PFN. By design, cluster 0 is always assigned to the first identified cluster, cluster 1 to the second, and so on. As the number of clusters increases, each subsequent cluster has access to less probability mass, since earlier clusters already occupy some of the posterior distribution. This is due to the Cluster-PFN performing full Bayesian inference across all clusters where each new cluster prediction is conditioned on the previously assigned clusters. As a result, the posterior probabilities for later clusters tend to be lower.

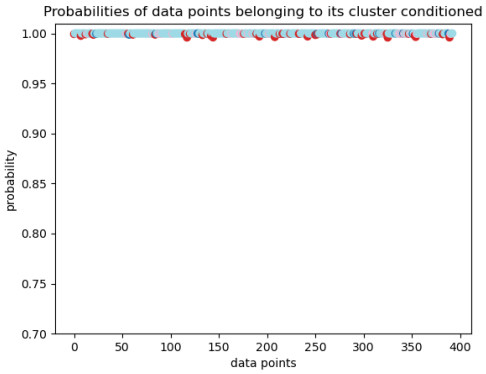
This issue can be mitigated by conditioning the Cluster-PFN on the total number of clusters. When the number of clusters is specified as input, the model can allocate its posterior probability mass more evenly across the expected clusters. In this conditioned setting, the posterior probabilities for all clusters become more balanced and reasonable. To implement this, we first allow the Cluster-PFN to generate its initial cluster predictions. We then feed the predicted number of clusters back into the model as a condition during a second inference forward pass to obtain the final label assignments.



(a) True data with cluster assignments: blue (cluster 0), red (cluster 1), pink (cluster 2), and cyan (cluster 3).



(b) Posterior probabilities of data points in dataset belonging to a cluster (unconditioned)



(c) Posterior probabilities of data points in dataset belonging to a cluster (conditioned)

Figure 5.8: Example of how conditioning improves probabilistic cluster assignments

Figure 5.8 illustrates the benefits of conditioning on the number of clusters. The original dataset is shown in Figure 5.8a, where the clusters are clearly separable. Without conditioning, the Cluster-PFN perfectly clusters the data points but assigns lower posterior probabilities to some of them, particularly cluster 2 (pink) and cluster 3 (cyan), as seen in Figure 5.8b. This occurs because earlier cluster assignments consume more of the posterior probability mass.

However, when conditioning is applied, the model redistributes the probabilities more evenly across all clusters. As shown in Figure 5.8c, the posterior assignments for all clusters are now much more balanced, demonstrating the effectiveness of conditioning in improving probabilistic cluster assignments. Therefore, when evaluating the external metrics of the Cluster-PFN

against the Bayesian GMM VI, we will use the conditioned Cluster-PFN.

5.3.2. Quantitative Comparison: Cluster-PFN vs. Bayesian GMM VI

We begin our evaluation by comparing the performance of Cluster-PFN and the Bayesian GMM VI on 30,000 synthetically generated datasets measuring ARI, AMI, purity, and NLL.

Model	AMI	ARI	Purity	NLL
2D Easy				
Cluster-PFN	**0.96 ± 0.09**	**0.98 ± 0.05**	0.97 ± 0.05	**0.15 ± 0.34**
Bayesian GMM VI	0.76 ± 0.29	0.87 ± 0.18	**0.98 ± 0.05**	1.06 ± 1.61
5D Easy				
Cluster-PFN	**0.97 ± 0.07**	**0.99 ± 0.04**	0.99 ± 0.04	**0.10 ± 0.27**
Bayesian GMM VI	0.92 ± 0.18	0.96 ± 0.11	0.99± 0.05	0.79 ± 1.60
2D Hard				
Cluster-PFN	**0.82± 0.18**	**0.93 ± 0.09**	0.92± 0.10	**0.47 ± 0.54**
Bayesian GMM VI	0.67 ± 0.28	0.84± 0.18	**0.93± 0.09**	1.40 ± 1.71
5D Hard				
Cluster-PFN	**0.89 ± 0.15**	**0.95 ± 0.08**	**0.95 ± 0.08**	**0.31 ± 0.43**
Bayesian GMM VI	0.83 ± 0.24	0.92± 0.15	0.94± 0.12	1.78 ± 2.76

Table 5.2: Average external clustering scores across 30,000 datasets for various experiments. Higher values indicate better performance for AMI, ARI, and purity, while lower values indicate better performance for NLL. Reported values are means ± standard deviations.

Table 5.2 presents the mean values and corresponding standard deviations for all external metrics, averaged across all evaluated datasets. As shown in the table, Cluster-PFN consistently outperforms Bayesian GMM VI in terms of ARI, AMI, and NLL. Cluster-PFN achieves higher average ARI and AMI scores with smaller standard deviations, as well as lower NLL with smaller standard deviation, indicating both better clustering performance and greater stability across datasets. While the purity scores of both methods are comparable, Bayesian GMM VI shows a slight advantage in this metric.

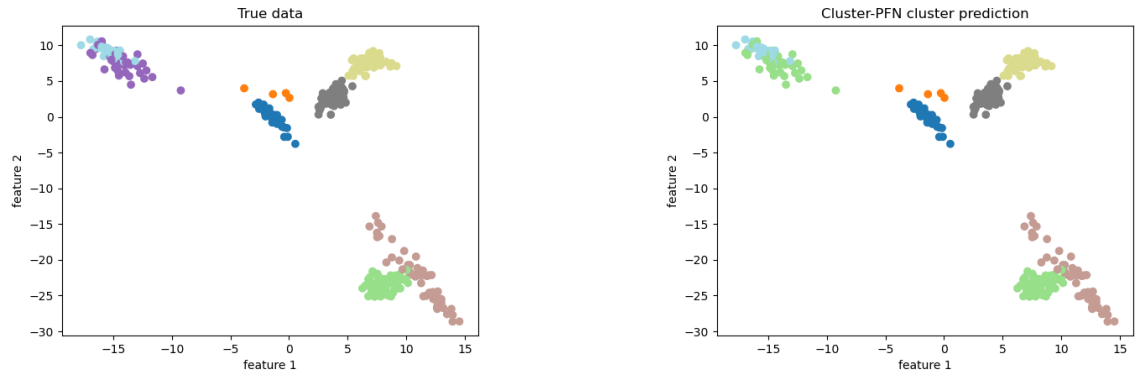
To better quantify the comparison between the models, we examine the number of datasets where each model outperforms the other across the various metrics on the same set of 30,000 datasets.

Model	AMI	ARI	Purity	NLL
2D Easy				
Cluster-PFN	**77.3**	**76.8**	11.5	**76.3**
Bayesian GMM VI	13.3	13.2	**26.5**	16.7
5D Easy				
Cluster-PFN	**44.4**	**43.5**	11.4	**49.3**
Bayesian GMM VI	18.2	16.4	**18.2**	28.9
2D Hard				
Cluster-PFN	**61.8**	**62.8**	17.4	**71.3**
Bayesian GMM VI	30.5	29.4	**48.7**	23.6
5D Hard				
Cluster-PFN	**41.4**	**43.9**	24.3	**59.7**
Bayesian GMM VI	35.8	32.5	**37.0**	27.8

Table 5.3: Win rate for each model on the external metrics (AMI, ARI, purity, and NLL) across 30,000 datasets for various experiments (ties not included).

Table 5.3 presents the win rates of each model across the different metrics. Similarly to Table 5.2 Cluster-PFN consistently outperforms Bayesian GMM VI in both AMI, ARI, and NLL across all settings.

One consistent observation is that the Cluster-PFN does not outperform the Bayesian GMM VI in terms of purity. A possible explanation for this behavior is illustrated below.



(a) True data

(b) Cluster-PFN prediction

Figure 5.9: Cluster-PFN predicting impure clusters

Figure 5.9 shows a sampled dataset, where the ground truth clustering is displayed in Figure 5.9a and the Cluster-PFN prediction is shown in Figure 5.9b. We observe that the Cluster-PFN assigns both the top-left and bottom-right regions to the same cluster. We hypothesize that the model interprets the two Gaussian blobs at the ends as a single elongated, correlated

Gaussian structure, and thus merges them into one cluster. Therefore the purity decreases because of the large mixtures.

5.3.3. Quantitative Comparison: Cluster-PFN vs. Bayesian GMM VI on Large Datasets

The comparisons of external metrics and NLL loss were conducted on datasets containing 100 to 500 data points. One of Cluster-PFN's strengths is that, although it was trained only on datasets of this size range, it can generalize to cluster much larger datasets. To evaluate this capability, we conducted an experiment by sampling 5,000 datasets, each containing 5,000 data points, and assessed Cluster-PFN's performance on these larger datasets against the Bayesian GMM VI.

Model	AMI	ARI	Purity	NLL
2D Easy				
Cluster-PFN	**66.9**	**67.1**	11.0	**73.7**
Bayesian GMM VI	23.9	23.2	**40.5**	22.3
5D Easy				
Cluster-PFN	25.2	24.3	11.0	35.6
Bayesian GMM VI	**31.4**	**28.8**	**28.6**	**41.0**
2D Hard				
Cluster-PFN	**49.0**	**50.1**	10.8	**60.0**
Bayesian GMM VI	45.2	44.0	**67.0**	36.6
5D Hard				
Cluster-PFN	23.8	28.9	24.4	**46.3**
Bayesian GMM VI	**56.0**	**50.6**	**50.5**	40.6

Table 5.4: Win rate for each model on the external metrics (AMI, ARI, purity, and NLL) across 5,000 large datasets for various experiments (ties not included).

Table 5.4 shows that Cluster-PFN achieves higher win rates in AMI, ARI, and NLL in the 2D cases. However, in the 5D cases, Bayesian GMM VI attains higher win rates in AMI and ARI, with comparable NLL performance. As before, Bayesian GMM VI consistently outperforms in purity scores. These results demonstrate that Cluster-PFN can generalize to larger datasets while maintaining strong performance within certain metrics.

5.3.4. Quantitative Comparison: Cluster-PFN vs. Bayesian GMM on NLL Loss Over Time

Since the Cluster-PFN offers very fast inference and efficient computation on batched inputs, we analyze the trade-off between NLL loss and computation time for both the Cluster-PFN and the Bayesian GMM VI. The Bayesian GMM VI relies on iterative optimization, where setting longer runtimes typically leads to better results. Comparing these two models highlights the trade-off between NLL performance and computational efficiency.

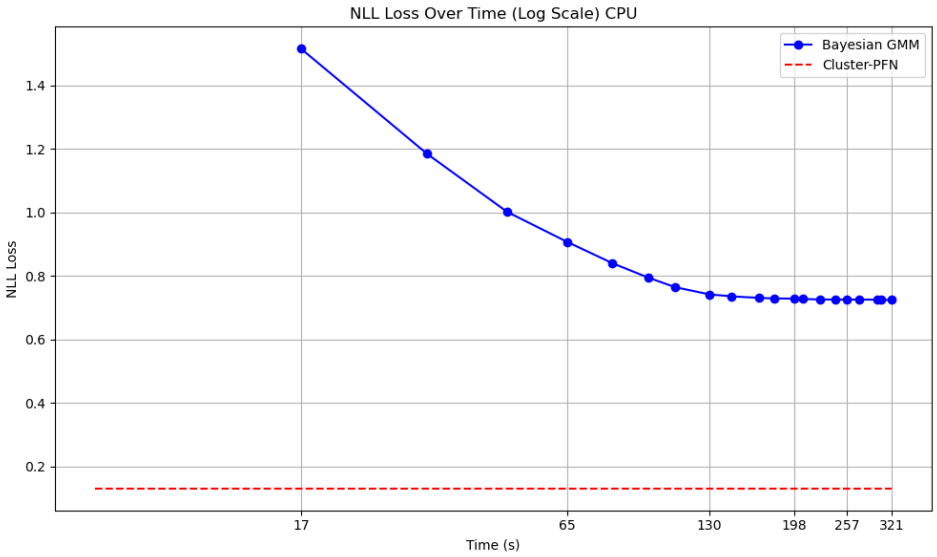


Figure 5.10: Comparison of average NLL loss over time across 10 datasets for Cluster-PFN and Bayesian GMM VI for 2D Easy experiment

Figure 5.10 shows the average NLL loss over time across 10 datasets for both Cluster-PFN and the Bayesian GMM VI. The Cluster-PFN achieves an average NLL loss of approximately 0.1 in under 10 seconds. In contrast, the Bayesian GMM VI starts with an NLL loss of around 1.5 and gradually converges to approximately 0.9 over time. These results highlight the superior speed and accuracy of Cluster-PFN compared to the Bayesian GMM VI.

5.4. RQ4: How does the Cluster-PFN perform against GMM and K-means++

We sampled 30,000 datasets and compared the performance of Cluster-PFN against the GMM and K-means++ models. To evaluate the models, we computed their average ranks, where a rank of 1 indicates the best performance and 3 the worst. Notably, the GMM and K-means++ models were given an advantage by providing them with the true number of clusters.

Model	2D Easy	5D Easy	2D Hard	5D Hard
Cluster-PFN	**1.57**	**1.57**	**1.60**	**1.56**
GMM	1.96	2.02	1.75	1.72
K-means++	2.47	2.41	2.65	2.72

Table 5.5: Mean AMI ranks of the models across 30,000 datasets for various experiments (lower is better).

Table 5.5 presents the average ranks of different models based on AMI scores across the evaluated datasets. Cluster-PFN consistently achieves the best average rank under all conditions, indicating superior clustering performance. GMM ranks second, while K-means++ performs the worst, particularly in the Hard setting. Similar trends are observed for ARI and purity metrics across the experiments. For corresponding tables of the additional metrics, please refer

to Appendix C.

6

Discussion and Future Work

This section outlines the main findings of the thesis, discusses the implications of the results, highlights the limitations of our approach, and outlines some of the future work.

6.1. Cluster-PFN and Bayesian Clustering

The experimental results demonstrate that Cluster-PFN can effectively perform Bayesian clustering when trained sufficiently on a prior. This suggests that in scenarios where prior knowledge about the types of clustering is available, Cluster-PFN can produce strong and reliable results. Furthermore, its conditioning mechanism enables users to choose clusterings that deviate from the original clustering. As a result, the model supports not just one-shot clustering, but flexible and interactive clustering under various constraints.

However, there are a few drawbacks to the construction of the data. The preprocessing step that deterministically assigns cluster labels based on distance from the origin may introduce unwanted dependencies. Furthermore, distances in very high dimensions become less informative and break down [8]. Therefore the Cluster-PFN would not be able to perform well in high dimensions.

6.2. Predicting Clusters

The cluster prediction results demonstrate that the Cluster-PFN outperforms other methods in estimating the number of clusters. This suggests that the model effectively captures the underlying data distribution.

In unsupervised learning settings, the true number of clusters is unknown and must be estimated. Having a model that can jointly cluster and predict the number of clusters removes the need for separate model selection procedures. Moreover, since Cluster-PFN can generalize the cluster prediction from training data, it is feasible to create a smaller, dedicated PFN variant that solely predicts cluster numbers. Such a lightweight model could serve as an accurate heuristic in determining the number of clusters.

There is an issue with how we compute the number of clusters predicted by the Bayesian GMM. Currently, we assign each data point to the cluster with the highest posterior probability and then count the number of unique clusters. This is essentially a heuristic approach. To ensure a fairer comparison, it would be better to use the full posterior probabilities from the Bayesian GMM and compare them directly to the cluster distributions produced by the Cluster-PFN.

6.3. Evaluating Clustering Algorithms

When evaluating the performance of Cluster-PFN against other clustering algorithms, we observe strong results on AMI and ARI, outperforming the Bayesian GMM VI, GMM, and K-means++. Additionally, Cluster-PFN achieves lower NLL loss compared to the Bayesian GMM VI. Notably, when trained on only 100–500 data points, Cluster-PFN is able to cluster datasets of 5,000 points at a level comparable to the Bayesian GMM VI. This demonstrates that the Cluster-PFN can scale well to large datasets. Furthermore, the inference speed of Cluster-PFN also outperforms the Bayesian GMM VI, demonstrating that Cluster-PFN is a viable option in terms of both accuracy and speed.

6.4. Future Work

Permutation invariant cluster assignments: An approach to mitigate the deterministic labeling of the data is to make the cluster assignments permutation invariant, thereby eliminating the need for fixed label assignments. This would also help address the high-dimensionality issue, as the data generation process would no longer rely on distance-based assignments. One possible strategy is to have Cluster-PFN predict the cluster means, where the space of possible means is discretized into predefined buckets. The model would then determine which data points correspond to which mean buckets. However, this approach may not scale well to high-dimensional data due to the exponential growth of discretization complexity. An alternative is to modify the loss function to incorporate a matching-based objective, such as bipartite matching which Carion et al. [11] have applied to the transformer architecture to avoid dealing with permutations.

Feature invariance: Currently, Cluster-PFN is permutation invariant with respect to the order of the samples. However, extending this invariance to the feature dimension could further improve performance, since the order of features should not affect the clustering results. Previous work has shown that model performance improves when incorporating feature invariance [24, 5]. We did not implement feature-wise attention due to the much longer training times.

Improving conditioning: The current conditioning mechanism performs well when the specified cluster counts are close to the Cluster-PFN's predicted number of clusters. However, it is less effective when conditioning on arbitrary cluster counts. Enhancing the model to handle a wider range of conditioned cluster counts would make it more flexible and broadly applicable. This could be achieved either by modifying the model architecture or by integrating the conditioning directly into the loss function, which may help enforce the condition more strongly during training.

Scaling up: Our experiments were conducted using a maximum of five input features, with the

model capped at predicting up to 10 clusters for both cluster assignments and the number of clusters. Scaling the model to handle more features and clusters can help validate its strengths or reveal potential pitfalls.

Exploring additional Bayesian clustering methods: In this thesis, we focused exclusively on model-based clustering algorithms. However, partition-based Bayesian clustering methods also exist, which rely on minimizing a defined loss function. These methods can generally be divided into three main categories: density-based approaches [6], hierarchical approaches [22, 28, 34], and distance-based approaches [14, 15, 16, 33]. Comparing Cluster-PFN to these methods could further validate its capabilities and highlight areas for improvement.

7

Conclusion

In conclusion, this thesis introduces Cluster-PFN, a Bayesian clustering model that leverages prior training and a conditioning mechanism to cluster datasets.

Our experiments show that Cluster-PFN effectively clusters datasets and can adapt to conditioning constraints when the specified cluster number is within a reasonable range of the model's own cluster predictions. Additionally, our results demonstrate that Cluster-PFN is highly accurate at determining the optimal number of clusters in a dataset, outperforming commonly used heuristics such as AIC, BIC, and the silhouette score.

Cluster-PFN outperforms Bayesian clustering algorithms like the Bayesian GMM VI in terms of ARI, AMI, and NLL loss, while also surpassing traditional clustering methods such as GMM and K-means++. We further demonstrate that Cluster-PFN can generalize effectively to much larger datasets, even when trained on significantly smaller ones, highlighting its practical value in unsupervised learning tasks. Furthermore, Cluster-PFN achieves very fast inference times while maintaining accurate cluster assignments.

Potential future improvements include making the model invariant to feature ordering, eliminating reliance on deterministic labeling, scaling up the PFN, and comparing it to additional Bayesian clustering methods.

References

[1] Serkan Akogul and Murat Erisoglu. “A Comparison of Information Criteria in Clustering Based on Mixture of Multivariate Normal Distributions”. In: *Mathematical and Computational Applications* 21.3 (2016). ISSN: 2297-8747. DOI: 10.3390/mca21030034. URL: https://www.mdpi.com/2297-8747/21/3/34.

[2] M Usman Akram et al. “Detection of neovascularization in retinal images using multivariate m-Mediods based classifier”. In: *Computerized Medical Imaging and Graphics* 37.5-6 (2013), pp. 346–357.

[3] Ekin Akyürek et al. *What learning algorithm is in-context learning? Investigations with linear models*. 2023. arXiv: 2211.15661 [cs.LG]. URL: https://arxiv.org/abs/2211.15661.

[4] Christophe Andrieu et al. “An Introduction to MCMC for Machine Learning”. In: *Machine Learning* 50 (Jan. 2003), pp. 5–43. DOI: 10.1023/A:1020281327116.

[5] Michael Arbel, David Salinas, and Frank Hutter. *EquiTabPFN: A Target-Permutation Equivariant Prior Fitted Networks*. 2025. arXiv: 2502.06684 [cs.LG]. URL: https://arxiv.org/abs/2502.06684.

[6] Raffaele Argiento, Andrea Cremaschi, and Alessandra Guglielmi. “A “Density-Based” Algorithm for Cluster Analysis Using Species Sampling Gaussian Mixture Models”. In: *Journal of Computational and Graphical Statistics* 23 (Oct. 2014). DOI: 10.1080/10618600.2013.856796.

[7] Michael Betancourt. *A Conceptual Introduction to Hamiltonian Monte Carlo*. 2018. arXiv: 1701.02434 [stat.ME]. URL: https://arxiv.org/abs/1701.02434.

[8] Kevin Beyer et al. “When Is “Nearest Neighbor” Meaningful?” In: *Database Theory — ICDT’99*. Ed. by Catriel Beeri and Peter Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 217–235. ISBN: 978-3-540-49257-3.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

[10] Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[11] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872.

[12] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways*. 2022. arXiv: 2204.02311 [cs.CL]. URL: https://arxiv.org/abs/2204.02311.

[13] Adrian Corduneanu and Christopher Bishop. “Variational Bayesian model selection for mixture distribution”. In: *Artificial Intelligence and Statistics* 18 (Jan. 2001), pp. 27–34.

[14] David B. Dahl, Jacob Andros, and J. Brandon Carter. *Cluster Analysis via Random Partition Distributions*. 2021. arXiv: 2106.02760 [stat.ME]. URL: https://arxiv.org/abs/2106.02760.

[15] David B. Dahl, Ryan Day, and Jerry W. Tsai and. “Random Partition Distribution Indexed by Pairwise Information”. In: *Journal of the American Statistical Association* 112.518 (2017). PMID: 29276318, pp. 721–732. DOI: 10.1080/01621459.2016.1165103. eprint: https://doi.org/10.1080/01621459.2016.1165103. URL: https://doi.org/10.1080/01621459.2016.1165103.

[16] Leo L. Duan and David B. Dunson. “Bayesian Distance Clustering”. In: *Journal of Machine Learning Research* 22.224 (2021), pp. 1–27. URL: http://jmlr.org/papers/v22/20-688.html.

[17] R. A. Fisher. *Iris*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56C76. 1936.

[18] Chris Fraley and Adrian E Raftery and. “Model-Based Clustering, Discriminant Analysis, and Density Estimation”. In: *Journal of the American Statistical Association* 97.458 (2002), pp. 611–631. DOI: 10.1198/016214502760047131. eprint: https://doi.org/10.1198/016214502760047131. URL: https://doi.org/10.1198/016214502760047131.

[19] Pasi Fränti and Sami Sieranoja. *K-means properties on six clustering benchmark datasets*. 2018. URL: http://cs.uef.fi/sipu/datasets/.

[20] Syliva Fruwirth-Schnatter, Gilles Celeux, and Christian Robert. “Handbook of mixture analysis”. In: 1 (2019).

[21] Shivam Garg et al. *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes*. 2023. arXiv: 2208.01066 [cs.CL]. URL: https://arxiv.org/abs/2208.01066.

[22] Katherine Heller and Zoubin Ghahramani. “Bayesian hierarchical clustering”. In: Jan. 2005, pp. 297–304. DOI: 10.1145/1102351.1102389.

[23] Matt Hoffman et al. *Stochastic Variational Inference*. 2013. arXiv: 1206.7051 [stat.ML]. URL: https://arxiv.org/abs/1206.7051.

[24] Noah Hollmann et al. “Accurate predictions on small data with a tabular foundation model”. In: *Nature* (Jan. 2025). DOI: 10.1038/s41586-024-08328-6. URL: https://www.nature.com/articles/s41586-024-08328-6.

[25] Lawrence Hubert and Phipps Arabie. “Comparing partitions”. en. In: *J. Classif.* 2.1 (Dec. 1985), pp. 193–218.

[26] Jeen Mary John, Olamilekan Shobayo, and Bayode Ogunleye. “An Exploration of Clustering Algorithms for Customer Segmentation in the UK Retail Market”. In: *Analytics* 2.4 (Oct. 2023), pp. 809–823. ISSN: 2813-2203. DOI: 10.3390/analytics2040042. URL: http://dx.doi.org/10.3390/analytics2040042.

[27] Michael Jordan et al. “An Introduction to Variational Methods for Graphical Models”. In: *Machine Learning* 37 (Jan. 1999), pp. 183–233. DOI: 10.1023/A:1007665907178.

[28] David A. Knowles and Zoubin Ghahramani. “Pitman Yor Diffusion Trees for Bayesian Hierarchical Clustering”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015), pp. 271–289. DOI: 10.1109/TPAMI.2014.2313115.

[29] Yan Meng and Xiyu Liu. “Application of K-means algorithm based on ant clustering algorithm in macroscopic planning of highway transportation hub”. In: *2007 First IEEE International Symposium on Information Technologies and Applications in Education*. IEEE. 2007, pp. 483–488.

[30] Samuel Müller et al. *Transformers Can Do Bayesian Inference*. 2024. arXiv: 2112.10510 [cs.LG]. URL: https://arxiv.org/abs/2112.10510.

[31] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

[32] Thomas Nagler. *Statistical Foundations of Prior-Data Fitted Networks*. 2023. arXiv: 2305.11097 [stat.ML]. URL: https://arxiv.org/abs/2305.11097.

[33] Abhinav Natarajan et al. “Cohesion and Repulsion in Bayesian Distance Clustering”. In: *Journal of the American Statistical Association* 119.546 (Apr. 2023), pp. 1374–1384. ISSN: 1537-274X. DOI: 10.1080/01621459.2023.2191821. URL: http://dx.doi.org/10.1080/01621459.2023.2191821.

[34] Radford Neal. “Density Modeling and Clustering Using Dirichlet Diffusion Trees”. In: *Bayesian Statistics* 7 (Jan. 2003), pp. 619–629.

[35] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[36] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[37] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[38] Simone Romano et al. *Adjusting for Chance Clustering Comparison Measures*. 2015. arXiv: 1512.01286 [stat.ML]. URL: https://arxiv.org/abs/1512.01286.

[39] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[40] Amna Waheed et al. “Hybrid features and mediods classification based robust segmentation of blood vessels”. In: *Journal of medical systems* 39 (2015), pp. 1–14.

[41] Martin Wainwright and Michael Jordan. “Graphical Models, Exponential Families, and Variational Inference”. In: *Foundations and Trends in Machine Learning* 1 (Jan. 2008), pp. 1–305. DOI: 10.1561/2200000001.

[42] Max Welling and Yee Teh. “Bayesian Learning via Stochastic Gradient Langevin Dynamics”. In: Jan. 2011, pp. 681–688.

[43] Yan Yang et al. “DBSCAN clustering algorithm applied to identify suspicious financial transactions”. In: *2014 International conference on cyber-enabled distributed computing and knowledge discovery*. IEEE. 2014, pp. 60–65.

A

Distributions

A.1. Dirichlet Distribution

The Dirichlet is a multivariate distribution, over K random variables $0 \leq \mu_k \leq 1$, $\sum_{k=1}^{K} = 1$. Denoting $\mu = (\mu_1, ..., \mu_k)^T$ and $\alpha = (\alpha_1, .., \alpha_n)^T$, we have

$$p(\pi) = \text{Dir}(\pi|\alpha_0) = C(\alpha_0) \prod_{k=1}^{K} \pi_k^{\alpha_0 - 1} \tag{A.1}$$

where $C(\alpha) = \frac{\Gamma(\hat{\alpha})}{\Gamma(\hat{\alpha}_1)...\Gamma(\hat{\alpha}_K)}$ and $\Gamma(\alpha) = \int t^{\alpha-1} e^{-t} dt$

The purpose of $C(\alpha)$ is that it serves as a normalization constant so the pdf integrates to 1.

A.2. Wishart Distribution

$$\mathcal{W}(\Lambda \mid W, \nu) = B(W, \nu) \, |\Lambda|^{(\nu - D - 1)/2} \exp\left(-\frac{1}{2}\text{Tr}(W^{-1}\Lambda)\right) \tag{A.2}$$

Where

$$B(W, \nu) = |W|^{-\nu/2} \left(2^{\nu D/2} \, \pi^{D(D-1)/4} \prod_{i=1}^{D} \Gamma\left(\frac{\nu + 1 - i}{2}\right)\right)^{-1} \tag{A.3}$$

Where W is a $D \times D$ symmetric, positive definite matrix. The parameter ν is called the number of degrees of freedom and is restricted to $v \geq D$.

B

Further Details of Experimental Setup

B.1. Architecture

The Cluster-PFN architecture is fully based on the PyTorch [36] library and utilizes the encoder component of the Transformer. It employs an embedding dimension of 256 and a hidden dimension of 512. The model uses 4 attention heads and consists of 4 encoder layers. Each encoder layer includes the following components:

1. **Multi-Head Self-Attention** with 4 attention heads
2. **Add & Layer Normalization** applied after the attention mechanism
3. **Position-wise Feedforward Network**, consisting of:
    - A linear layer: $256 \to 512$
    - ReLU activation
    - A linear layer: $512 \to 256$
4. **Add & Layer Normalization** applied after the feedforward network

After passing through the encoder, a final layer maps each input to 10 output values. Each output represents the probability of the data point belonging to a specific cluster.

We use a cosine annealing learning rate schedule with a warm-up, applied to an AdamW optimizer initialized with a learning rate of 0.001.

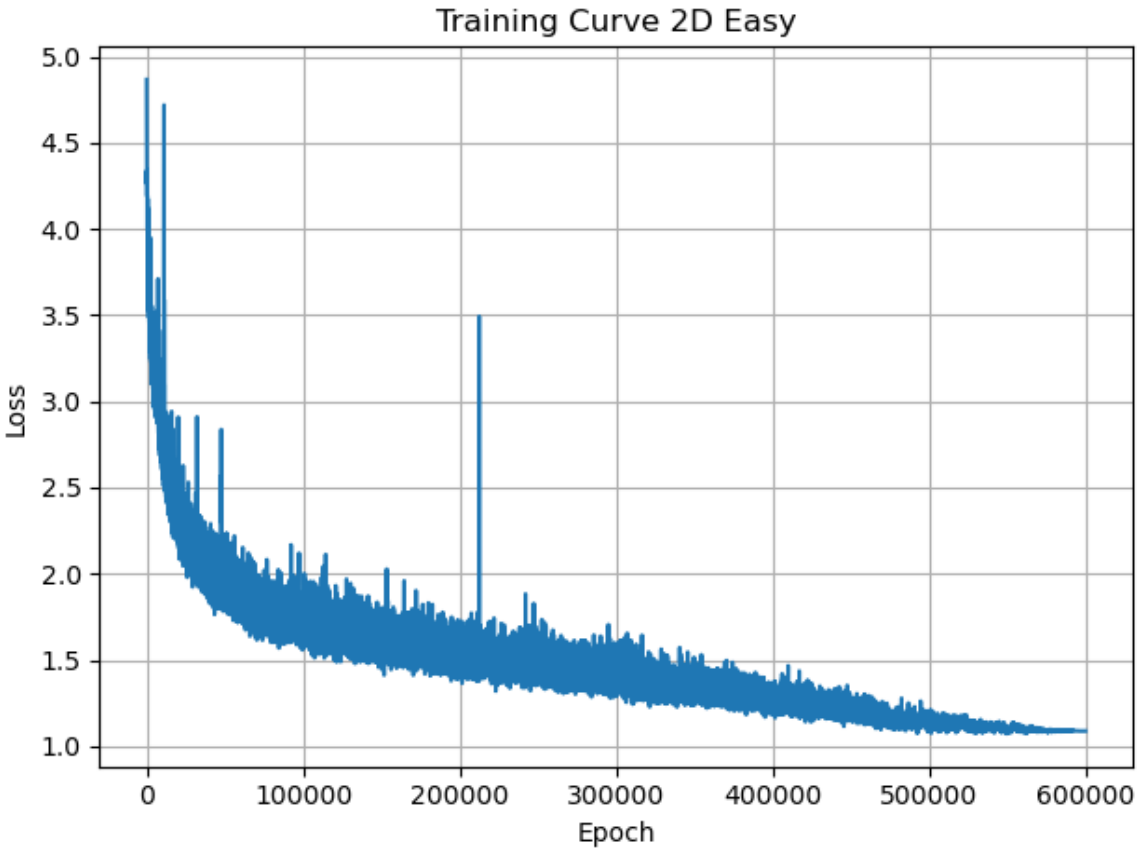
B.2. Loss Function

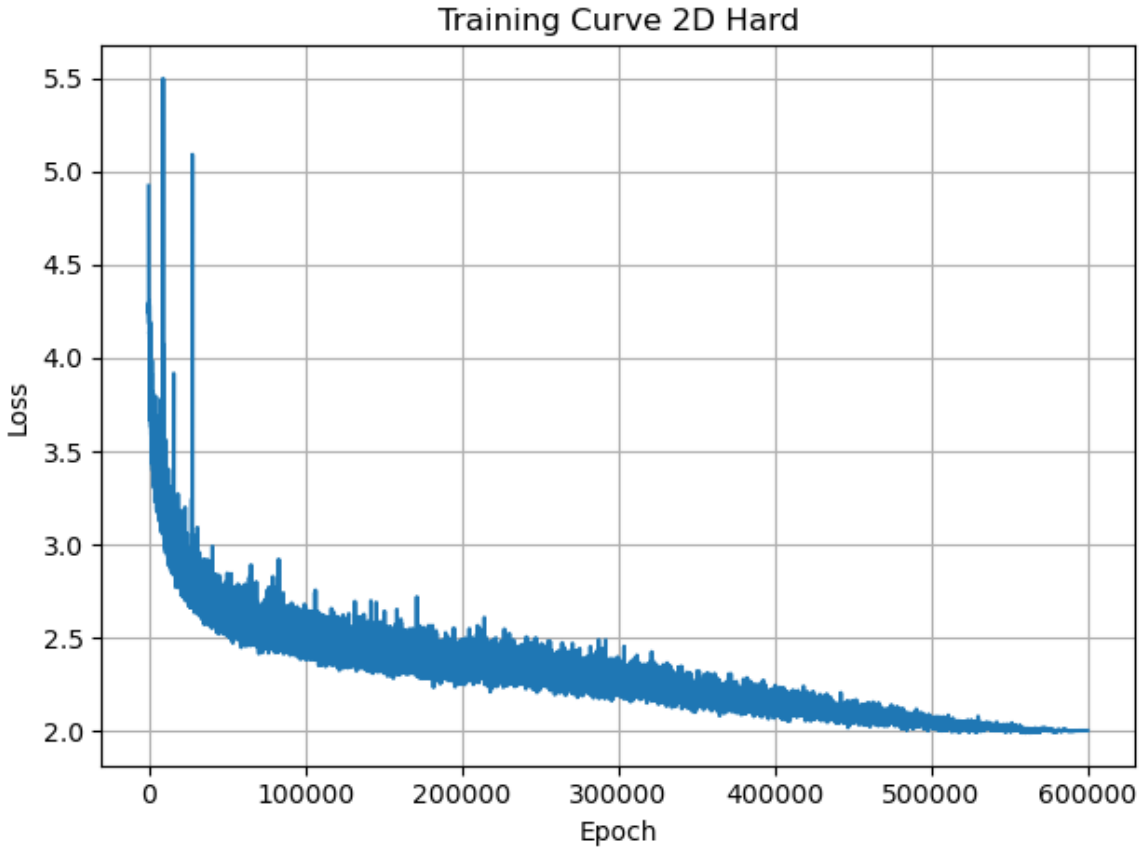
The loss function used is the cross-entropy loss. There are two components to the overall loss: the loss from the data point cluster assignments and the loss from the predicted number of clusters. The final loss is computed as the sum of the average cluster assignment loss and the cluster prediction loss.

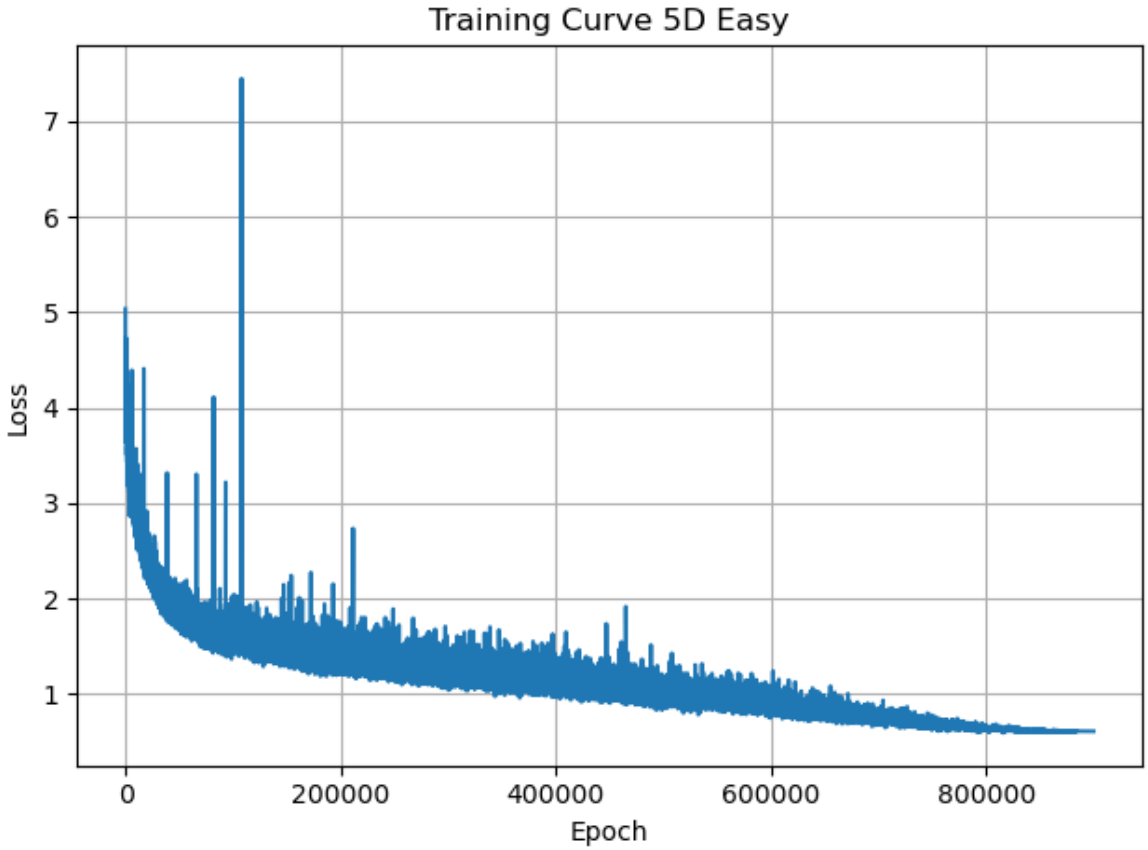
B.3. Training procedure

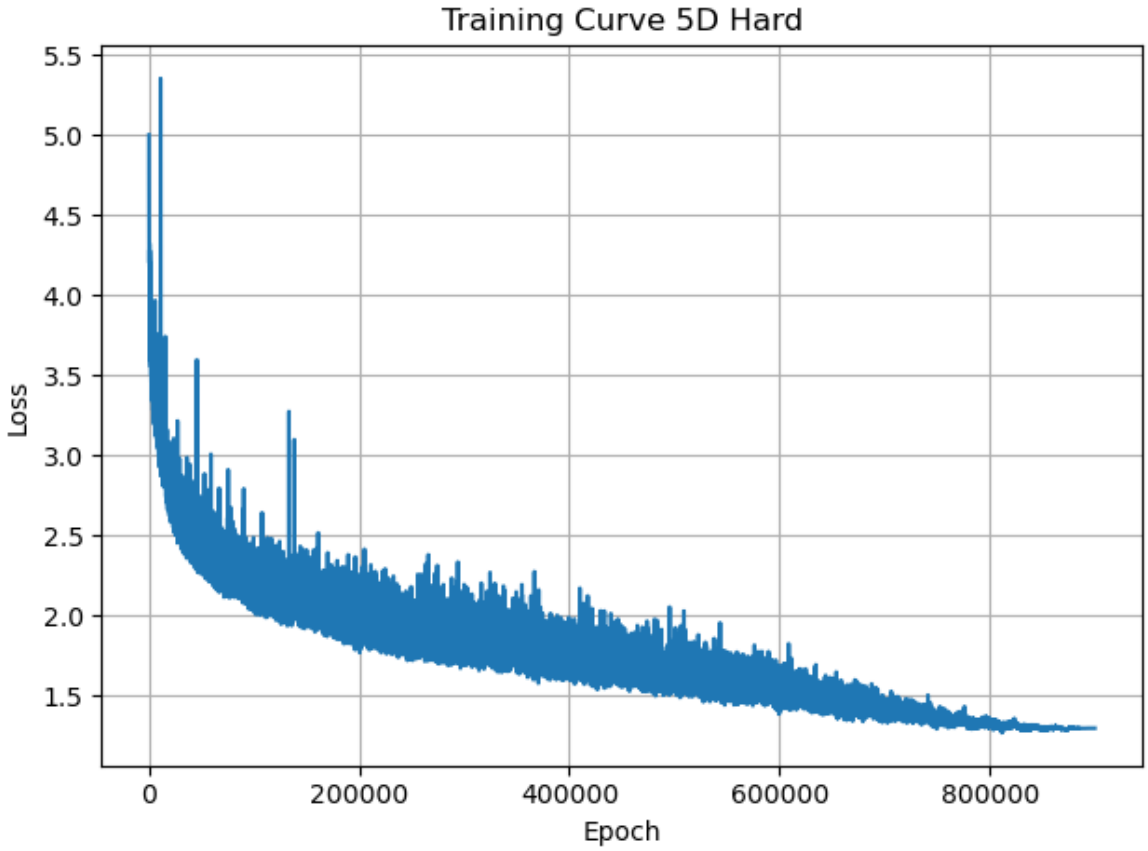
For each epoch, we generate a batch consisting of 100 datasets. Additionally, we fixed a validation set of 200 datasets, which is used to monitor the training curves of the model.

B.4. Training curves of different experiments




Training Curve 2D Hard
Loss
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
0
100000
200000
300000
400000
500000
600000
Epoch


Training Curve 5D Easy
Loss
Epoch
7
6
5
4
3
2
1
0
200000
400000
600000
800000


Training Curve 5D Hard
Loss
Epoch
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0
200000
400000
600000
800000

C

Additional Experimental Results

C.1. Rankings Cluster-PFN vs GMM vs KMeans++ on different metrics

Model	2D Easy	5D Easy	2D Hard	5D Hard
Cluster-PFN	**1.54**	**1.54**	**1.50**	**1.49**
GMM	1.99	2.05	1.82	1.79
K-means++	2.47	2.41	2.68	2.72

Table C.1: Mean ARI ranks of the models across 30,000 datasets for various experiments (lower is better).

Model	2D Easy	5D Easy	2D Hard	5D Hard
Cluster-PFN	**1.71**	**1.70**	**1.70**	**1.66**
GMM	1.93	1.99	1.72	1.70
K-means++	2.36	2.31	2.58	2.64

Table C.2: Mean Purity ranks of the models across 30,000 datasets for various experiments (lower is better).