Sample-Based t-SNE Embeddings

How different Sampling Strategies influence the Quality of Low-Dimensional Embeddings

Em Ketterer[1]

Supervisors: Klaus Hildebrandt[1]**, Martin Skrodzki**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Em Ketterer
Final project course: CSE3000 Research Project
Thesis committee: Klaus Hildebrandt, Martin Skrodzki, Cristoph Lofi

Abstract

Data visualisation is an important area of research: as the amount of data keeps increasing, we have to find ways of showcasing this data to provide an intuition for trends and patterns within it. This can be a particular challenge for high-dimensional data, since we cannot perceive it as is. A common approach is to use dimensionality-reduction techniques to bring the high-dimensional data into lower dimensions, which can then be visualised. One such technique is t-distributed Stochastic Neighbour Embedding (t-SNE), which produces good visualisations but struggles with long runtimes. This paper explores the effect of using sampled data instead of the full dataset to produce t-SNE embeddings, reducing the runtime of the algorithm and hence providing visualisations faster. We show that both visually and numerically, uniform random sampling and Poisson disk sampling can result in much faster runtimes while producing similar, or even more meaningful embeddings than the embedding of the entire dataset.

1 Introduction

The amount of data present in the world has been consistently increasing: in 2023, the volume of data on the Internet was about 123 zetabytes, a number predicted to grow to 394 zetabytes by 2028 [1]. With this sheer amount of data, it is vital that methods for data visualisation exist to help showcase clusters, patterns and outliers in the data [2], providing an intuition for individuals collecting and analysing said data.

For high-dimensional data, visualisation is more challenging since we cannot perceive more than three dimensions at once. A common solution is to apply dimensionality reduction algorithms to the dataset. Multiple such algorithms exist, one of which is t-distributed Stochastic Neighbour Embedding (t-SNE): a technique that associates each data point in the high-dimensional dataset with a two or three dimensional point [3]. This is done by first initializing a low-dimensional data embedding. Data points in this embedding are given a distribution according to their neighbours, and then their locations are updated to make the distribution as similar as possible to the high-dimensional data point distribution.

While t-SNE works very well for data visualisation [4], it is also rather slow, with a runtime complexity of $O(n^2)$ per iteration [5]. Many variations of the algorithm exist, using Fourier transforms or other algorithmic improvements to speed-up computation time, but overall, t-SNE remains time intensive for large datasets. In this paper, we investigate the effect of using a sample of the dataset to compute t-SNE embeddings instead. Such samples would be of smaller size, thus having a less extensive runtime which could be useful when approximate data visualisations are sufficient.

To investigate this, we compute embeddings of the dataset using different sampling rates and perplexity values. The quality of the resulting embeddings are compared visually and numerically using the area under the precision-recall curve as a measure. We investigate four sampling methods in this manner: uniform random sampling, farthest point sampling, Poisson disk sampling, and random walk sampling.

The results of this experimentation show that different sampling algorithms have vastly different effects. Notably, uniform random sampling creates similar sampled embeddings compared to the full dataset embedding. On the other hand, Poisson disk sampling performs the best numerically, and even preserves more information about the dataset through sub-clustering than present in the full embedding. With these results, we show that using sampling algorithms can have advantages over simply computing the full embedding.

2 Background

2.1 Dimensionality reduction using t-SNE

There are many methods for dimensionality reduction which can broadly be categorized into two types: linear and non-linear. Non-linear approaches create lower dimensional embeddings that preserve non-linear structures in the dataset, producing more meaningful representations [4]. While many other well-performing techniques exist for visualising high-dimensional data, they often create less representative embeddings [3]. T-SNE is a non-linear approach with the specific goal of creating good visualisations [3]. Comparing it to other high-dimensional data visualisation techniques reveals t-SNE to be one of the "two most effective methods for preserving local distance relationships among cells and providing informative visualisations" [4]. Therefore, we will focus on the t-SNE algorithm, since it is highly relevant for data visualisation.

The t-SNE algorithm functions by giving high-dimensional data $D = \{x_1, x_2, \dots, x_n\} \subseteq \mathbb{R}^d$ a distribution based on the pairwise probabilities, p_{ij}, between data points x_i and x_j:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq l} \exp(-||x_k - x_l||^2/2\sigma_i^2)}, p_{ij} = \frac{p_{j|i} + p_{i|j}}{2} \quad (1)$$

where the probability of a point given itself, $p_{i|i}$, is set to 0, and σ_i^2 represents the bandwidth of the Gaussian distribution around point x_i [3]. This bandwidth is usually smaller for dense parts of data, and in practice is defined such that the perplexity of the data matches the user-defined perplexity [5]. This user-defined perplexity, $Perp$, is a hyper-parameter for creating the high-dimensional distribution, roughly describing "the number of close neighbors each point has" [6].

$$Perp = 2^{-\sum_j p_{j|i} \log(p_{j|i})} \quad (2)$$

The high-dimensional data points are embedded into a lower dimensional representation, such that high-dimensional points x_i and x_j map to low-dimensional points y_i and y_j respectively. The low-dimensional points are then given a distribution defined through pairwise probabilities shown in Equation 3.

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}} \quad (3)$$

An initial embedding is often computed through other dimensionality reduction techniques, such as Principal Component Analysis (PCA) [7], a linear technique that finds the data's intrinsic dimensions ordered by variance, and uses this to reduce the data to the specified number of dimensions [4]. The initial embedding is then iteratively changed to make the low-dimensional distribution Q as similar as possible to the high-dimensional distribution P. This is done by minimising the Kullback-Leibler divergence of the two distributions, given in Equation 4, using gradient descent.

$$KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{4}$$

By using this method, data can be mapped to two or three dimensions while maintaining the close neighbours of the original data.

2.2 Sampling algorithms

Sampling is a useful approach applied in multiple areas of research within Computer Science to generate a representative subset of data that is easier to process. There are many sampling algorithms that can be used for this. While simpler algorithms usually have faster runtime, more complex algorithms aim to maintain more of the dataset's variance. We will consider a few different sampling algorithms with different strengths to see which produces better t-SNE embeddings.

Farthest Point Sampling (FPS)

FPS is an approach that iteratively generates a sample from the original dataset. At every step, the sample is extended with the data point farthest away from all previous samples, where the initial sample is randomly chosen [8]. This technique promises to preserve the spatial characteristics of the dataset, and is commonly used in Computer Graphics [9].

The main disadvantage of the standard FPS algorithm is its slow runtime [8]. At every iteration, the distance between the sampled point(s) and all other data points is computed. To reduce runtime, faster implementations exist [8] [9] [10]. The state of the art implementation of FPS is called QuickFPS [9]. It makes use of an auxiliary data structure: a KD-tree used to partition the dataset into buckets [8]. Distances to the sampled points are only calculated for points within a bucket if certain conditions are fulfilled, creating a significant reduction in computation and memory access time [8]. Using this improved algorithm decreases runtime, making it more applicable for large datasets [8]. Even though KD-trees become less efficient in higher dimensions [11], using the algorithm still provides a speed up of over 10 times for 50 dimensional data because of reduced and stored computations. Due to FPS's promise of preserving spatial characteristics, it is a very interesting algorithm to test.

Poisson Disk Sampling (PDS)

PDS is a technique that generates samples according to some probability distribution while ensuring every sample maintains some minimum distance to all other samples [12]. Due to the minimum distance, this approach avoids clusters of samples often found as a result of using other techniques, like uniform random sampling.

The most straightforward implementation of this algorithm is called dart-throwing; a randomly selected data point is either accepted or rejected as a sample depending on whether it maintains the minimum distance to all previous samples [13]. This algorithm becomes very slow towards the end of sample generation since most samples are rejected at this point [13].

Many improvements upon this implementation exist [14]. For an application using data points instead of linear sampling over an area, an elimination based technique is especially effective [15]. This algorithm assigns weights to each data point based on the distance to it's neighbours [15]. The data point with the highest weight is removed from the samples, causing all other weights to be updated, until only the desired amount of samples remain [15]. This approach functions well in high dimensions, allows for progressive sampling, and has a more efficient runtime than dart-throwing [15]. Using this algorithm will be interesting, because the spread of samples should be more even over the entire dataset, potentially changing the properties displayed by the t-SNE embeddings.

Random Walk Sampling (RWS)

In RWS, transition probabilities between all data points are defined. An initial sample is randomly chosen, after which the possible transitions are followed to generate the full sample. Methods based on this approach are very common for graph sampling since it produces unbiased samples [16].

Before beginning to sample, the transition probabilities need to be defined. In prior work, the probability of transitioning from data point x_i to x_j is defined as

$$p_{ij} = \frac{\exp(||x_i - x_j||^2/\sigma_i)}{\sum_k \exp(||x_i - x_k||^2/\sigma_i)} \tag{5}$$

where σ_i is chosen to ensure $\sum_j p_{ij}$ has a perplexity of one third the size of the dataset [17].

Using these defined probabilities, there are multiple algorithms that can be used to actually implement the random walks. For example, Metropolis-Hastings can be used to either accept or reject a new sample based on some probability distribution [16]. Another option is to start multiple random walks at so called landmarks: points identified to be important features of the dataset [17]. From there, a set number of steps is taken, and the final data point of the walk is noted down. If a data point is the end of enough random walks, it is sampled [17]. By starting multiple walks at different data points, unwanted bias is avoided. This algorithm is particularly interesting, since it is used in an adjacent context: creating interactive t-SNE data embeddings. Using it for RWS could reveal more about the internal distances of the dataset.

3 Related Work

The runtime complexity of the standard t-SNE algorithm is $O(n^2)$ per iteration [5], which does not scale well for large datasets. As a result, faster algorithms exist such as the Barnes-Hut-SNE: using the Barnes-Hut algorithm for approximating the gradient of the Kullback-Leibler divergence [18]. Due to this, the overall runtime complexity decreases to $O(n \log n)$ [18]. Another algorithm, called PM-t-SNE, uses fast Fourier transforms to approximate the gradients, leading

to slightly better speed-ups than the Barnes-Hut variant [19]. While these algorithms offer some speed-ups, they concern themselves with the entire dataset, with no research addressing the use of sampling instead.

Sampling strategies are used in adjacent research related to data visualisation using t-SNE. Prior work shows that a uniform sample can be used to estimate a good value for the perplexity hyper-parameter [5]. Another use-case is for a t-SNE variation called hierarchical t-SNE, where random walks are used to select different hierarchical levels of samples from the dataset, allowing fine and coarse representations of the same data to be calculated more quickly [17]. While both of these use sampling to improve the t-SNE workflow, neither analyse, in detail, how well the employed sampling strategy actually preserves the local data structure.

This shows that work has gone into improving t-SNE, even by using sampling, but no investigation into the actual quality of the embedding of samples has been done. Additionally, no prior work has investigated the impact of using different sampling strategies. This paper aims to fill this knowledge gap, and investigate how well differently sampled data can visualise high-dimensional datasets using t-SNE.

4 Methodology

To experiment with the quality of t-SNE embeddings under different sampling approaches, we use a Python repository[1]. Python has many libraries built for data analysis, making it a very useful programming language for this use-case. The library openTSNE[2] is especially useful, since it provides functionality to compute t-SNE embeddings, even through speed-ups using Fourier transforms, making it viable to use on larger datasets. Additionally, using Python makes it possible to build on existing code from [5], ensuring the experimentation does not have to be built from scratch.

To compute the t-SNE embeddings, we use the program described in Algorithm 1. Here, the chosen dataset is loaded, given an initial embedding, and reduced to 50 dimensions if it exceeds this dimensionality. This reduction in dimensions compresses the initial data, an approach shown to speed up computation without strongly impacting the quality of the results [3]. If a sample strategy is provided, the dataset and initial embedding is sampled using this sampling strategy. We then use the (sampled) data to compute the low-dimensional embedding of the dataset.

The sampling methods are each implemented separately. For FPS, the algorithm of QuickFPS is implemented since this makes the sampling strategy relatively fast to work with [8]. PDS is implemented using the elimination based technique, since this does not require determining an arbitrary minimum distance between data points [15]. Weights for each data point are computed as the sum of distances to its high-dimensional neighbours, as determined by the used perplexity parameter. RWS is implemented using a variation of the algorithm from the hierarchical t-SNE paper: 10 random walks, each of length 30, are started at every data point [17]. The data points at the end of the most random walks are then sampled. We chose this variation of the algorithm to ensure that runtimes would remain below 3 hours.

Algorithm 1 Computing t-SNE Embeddings

```
data, labels ← load dataset
initial_embedding ← PCA of data to 2 dimensions
sample_strat ∈ {None, Uniform, FPS, PDS, RWS}
if dimensionality of data > 50 then
    data ← PCA of data to 50 dimensions
if sample_strategy ≠ None then
    data ← sample of dataset using sample_strat
    initial_embedding ← sample of initial embedding
affinities ← compute affinities of data
embedding ← embed data using fast Fourier t-SNE
return embedding
```

To analyse the quality of embeddings, we use two comparison methods. The first is to visually compare the embeddings made per sampling strategy using different sampling rates and perplexity values. The exact values that are tested can be seen in Table 1. Note that values within the same column are linearly related to each other based on the sample size of the dataset. Using this visualisation gives a good impression of how well the embedding of the dataset works. The other chosen comparison method numerically analyses the embeddings by using the area under the precision-recall curves. Here, *precision* represents the average fraction of low-dimensional neighbours that are correctly classified as neighbours. *Recall*, on the other hand, represents the average fraction of high-dimensional neighbours that remain neighbours in the lower dimensions. Using this method allows for a quantitative comparison, with bigger areas representing better embeddings and an area of one being a perfect representation [20]. Using these two methods, it is possible to comprehensively analyse the quality of different embeddings.

In addition to this, we run an experiment concerning the runtime of the sampling algorithms. Using this together with an analysis on how much time is saved by using a sample of the dataset to create the t-SNE embeddings makes it possible to numerically assess the speed-up gained by using a sampling-based approach.

Lastly, it is important to note that all analysis is run on at least two datasets: the MNIST dataset and the C.Elegans dataset. The MNIST dataset is a set of 70,000 images of handwritten digits[3], and C.Elegans is a dataset of 89,701 entries containing single cell and single nucleus RNA sequencing data[4]. Additionally, we use a third dataset when visualising embeddings for uniform random sampling, FPS and PDS, being the Wong dataset. This dataset contains 327,457 data points on "lymphocyte composition across human tissues"[5]. Unfortunately, using this dataset for more analysis is outside the scope of this research due to long runtimes. By using two, and sometimes three datasets of vastly different origin, we show that the results of this paper are broadly applicable, and not a mere coincidence for a specific dataset.

[1] https://github.com/WinterStormStyx/sampling-t-SNE

[2] https://opentsne.readthedocs.io/en/stable/

[3] http://yann.lecun.com/exdb/mnist/

[4] https://github.com/Munfred/wormcells-data/releases

[5] http://flowrepository.org/id/FR-FCM-ZZTM

Sample Rate	Perplexity Values per Dataset											
	MNIST				C.Elegans				Wong			
0.1		7	21	144	3	12	34	105	3	30	70	100
0.4	4	28	84	576	12	48	136	420	12	120	280	400
0.7	7	49	147	1008	21	84	238	735	21	210	490	700
1.0	10	70	210	1440	31	121	341	1051	30	300	700	1000
	$\frac{1M}{7000}$	$\frac{7M}{7000}$	$\frac{21M}{7000}$	$\frac{144M}{7000}$	$\frac{3M}{8970}$	$\frac{12M}{8970}$	$\frac{34M}{8970}$	$\frac{105M}{8970}$	$\frac{3M}{32745}$	$\frac{30M}{32745}$	$\frac{70M}{32745}$	$\frac{100M}{32745}$

Table 1: The perplexity values chosen to visualise per sampling rate for each dataset. These values are linearly related within a column. The relationship is shown in the bottom row of the table, where M represents the sample size = sample rate × dataset size.

5 Results

5.1 Visual evaluation of sampling strategies

To visually evaluate the resulting embeddings, graphs with different parameters are plotted. For each sampling rate, different perplexity values are chosen based on the dataset. These values are drawn from experimentation in [5], and can be seen in Table 1. By using three datasets as well as a variety of sampling and perplexity values, a thorough visual insight into the embeddings can be gained. Note that the figures for uniform random sampling in this section are recreations from [5].

The MNIST dataset

The first dataset we discuss is the MNIST dataset, consisting of 70,000 handwritten numbers. Using this dataset, we test the visual quality of embeddings given different sample sizes and perplexity values, as can be seen in Figure 1. Here, embeddings produced with different sampling strategies are positioned next to each other to allow for easier comparison.

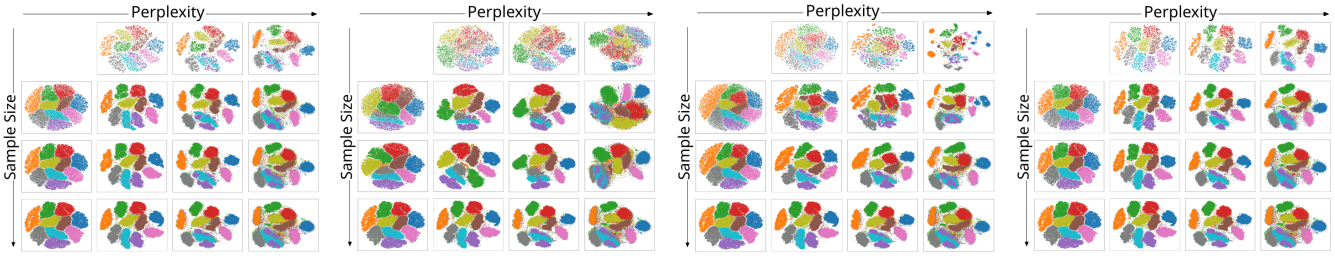
The first sampling strategy that we tested is uniform random sampling, a sampling strategy where a specified number of data points are randomly selected from the dataset. The produced embeddings can be seen in Figure 1a. This figure demonstrates that the resulting embeddings using this sampling strategy are quite different for different perplexity values (different column), but stay relatively consistent within the same column, meaning they are consistent when the same perplexity ratio is used. The consistency within each column implies that a similar embedding is achieved when a sample of the dataset is used instead of the full dataset, suggesting that the sampled embeddings are a rather good visualisation.

FPS is another strategy that we can use to obtain a sample of the original dataset. To showcase the quality of embeddings, the same figure as for uniform random sampling is created, shown in Figure 1b. While similar patterns seem to be preserved, it is immediately noticeable that the sampled embeddings are not as faithful to the full dataset embedding (shown in the lowest row of the figure) as is the case for uniform random sampling.

Upon closer inspection, Figure 1b seems to have one distinct difference to Figure 1a, being the complete lack of an orange cluster in most of the sampled embeddings. Through a separate experiment, we found that no matter what label the initially picked point is from, the data points belonging to the "1" (orange) cluster remains incredibly undersampled. This is shown in Table 2, presenting the number of samples selected from each label depending on what label the initial data point is randomly chosen from. We suspect that this undersampling is the cause of the visually worse embeddings.

Initial Label Chosen	Number of Samples selected from each Label									
	0	**1**	2	3	4	5	6	7	8	9
0	52.8	**4.0**	99.5	62.8	40.4	58.6	40.4	23.0	92.3	26.2
1	53.6	**4.7**	100.3	61.1	39.1	58.9	39.3	23.7	93.3	26.0
2	53.8	**3.9**	102.3	61.0	40.2	57.6	40.5	22.9	92.3	25.5
3	56.0	**3.6**	99.9	59.8	40.5	60.6	40.1	22.1	94.2	23.2
4	54.3	**4.7**	99.0	61.2	41.2	58.3	41.5	23.2	91.1	25.5
5	55.2	**3.7**	102.0	60.3	39.3	59.3	39.6	23.3	91.1	25.7
6	53.4	**3.4**	103.0	59.5	40.2	56.6	39.3	23.1	95.2	26.3
7	55.6	**3.9**	100.5	61.5	40.9	57.6	39.7	23.1	91.5	25.7
8	55.6	**3.9**	104.2	61.1	41.1	58.2	28.6	22.5	89.0	25.8
9	55.5	**3.1**	102.0	58.8	39.4	58.2	39.9	22.2	94.0	26.9

Table 2: The average number of samples selected from each of the MNIST labels after being given a random, initially chosen sample from one of the 10 labels. In total, 10 trials are run for each initial label, and during each run 500 samples are selected.



(a) Uniform Random Sampling (b) FPS (c) PDS (d) RWS

Figure 1: The impact of sample size and perplexity on embeddings of the MNIST dataset using different sampling strategies.

PDS seems to produce relatively consistent embeddings within the same column, as shown in Figure 1c. However, it is noticeable that more clusters have merged than for uniform random sampling. This makes the embedding less clear in terms of separating differently labelled data points. Additionally, PDS seems to produce some splitting of clusters at low sample sizes and high perplexity values. Investigating this reveals that this sub-clustering is not random, and instead seems to be preserving more information about the dataset. For example, the cluster of 1's is split based on how tilted the number is written. Some exemplary data points for these sub-clusters are shown in Figure 3, demonstrating the properties being preserved by this sub-clustering of the embedding using the highest perplexity and lowest sample size. This result shows that using a sample found with PDS actually preserves more information about the dataset than in the full embedding.

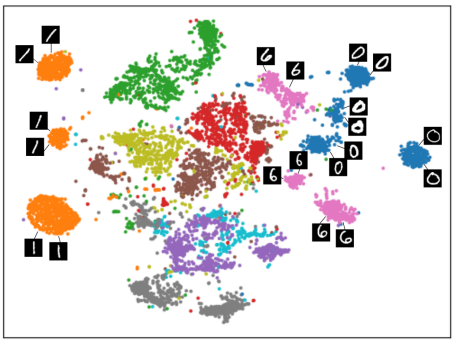


Figure 3: Visualisation of the sub-clustering for highest perplexity, lowest sample size of the MNIST dataset using PDS with exemplary data points.

RWS is the last sampling algorithm we consider the visual quality of embeddings for. Taking a look at these, it appears that the resulting embeddings are pretty consistent within a column. In fact, the results, shown in Figure 1d, seem really similar to those obtained by uniform random sampling (see Figure 1a). The only noticeable difference is that clusters appear slightly more separated for the smallest sample size. This similarity might be because the MNIST dataset has a similar number of data points for each of the labels in the dataset, and so the random walks converge to uniform random sampling, since all data points have similar probabilities.

The C.Elegans dataset

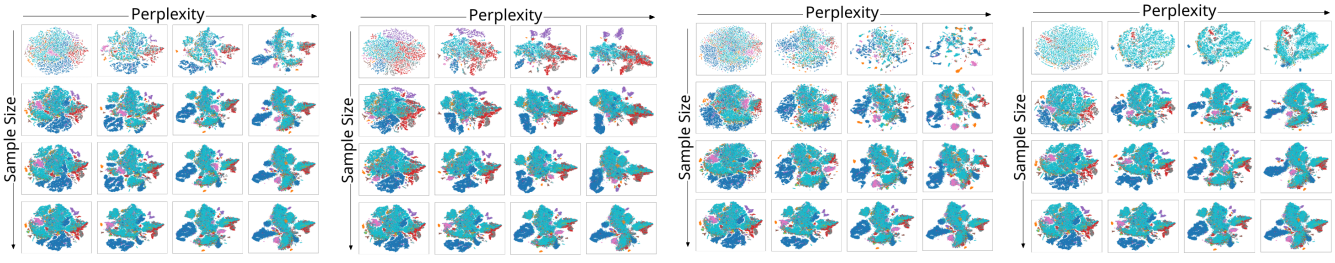
To test whether the presented results remain consistent over multiple datasets, the same visuals are reproduced for a second dataset. Here, we present the results using the C.Elegans dataset, containing 89,701 data points on single cell and single nucleus RNA sequencing data.

Uniform random sampling seems to, once again, produce visually consistent embeddings within the same column. This can be seen in Figure 2a. The still present consistency supports our previous findings: a sampled part of the dataset can be used to produce an embedding which is representative of the entire datasets embedding.

Calculating the same figure using FPS, shown in Figure 2b, produces much more promising results than using the MNIST dataset. While the embeddings are not the most consistent within a column, all clusters seem to at least be present. However, there still seems to be some bias, resulting in more weight being given to groupings like the purple cluster. While this changes the visual of the resulting embeddings for smaller sample sizes, it might be a positive change since this causes clusters in the dataset to be visualised clearly instead of disappearing almost entirely like happens using uniform random sampling. In general, it appears that while these embeddings are less visually consistent than uniform random sampling, they still maintain some patterns and even cause smaller clusters to be visualised instead of disappearing.

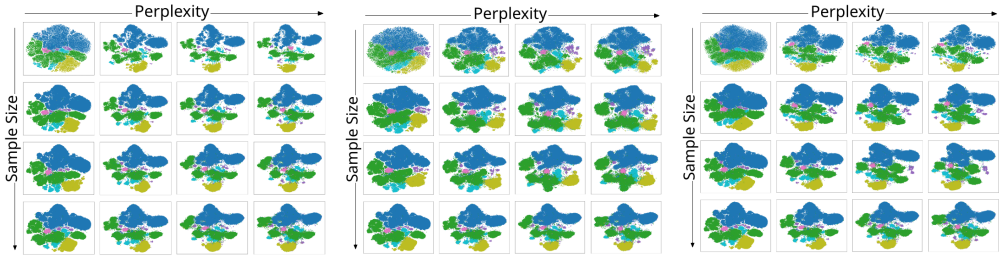
The embedding of the dataset using PDS, shown in Figure 2c, once again produces visually consistent results with some splitting of clusters occurring at low sample sizes and high perplexity values. While there is no easy way to check whether the sub-clustering is meaningful for this dataset, the fact that the results are so similar to those of the MNIST dataset imply that it could also be useful in this instance.

Using RWS produces embeddings which, in contrast to the MNIST dataset, actually differ from those obtained through uniform random sampling. In these embeddings, shown in Figure 2d, the results seem pretty consistent within a column, the only exception being for the lowest sample size where there seems to be a heavy bias favouring the light cyan cluster; other clusters are barely present in the embedding. At least visually, this does not seem ideal since a lot of information on the number of clusters and different aspects present in the dataset is lost.



(a) Uniform Random Sampling (b) FPS (c) PDS (d) RWS

Figure 2: The impact of sample size and perplexity on embeddings of the C.Elegans dataset using different sampling strategies.



(a) Uniform Random Sampling (b) FPS (c) PDS

Figure 4: The impact of sample size and perplexity on embeddings of the Wong dataset using different sampling strategies.

The Wong dataset

To test whether the results hold on an even more general basis, embeddings are created for a third dataset: the Wong dataset containing 327,457 data points. The produced embeddings can be seen in Figure 4. Due to long runtimes, it is not possible to test RWS on this dataset.

Once again, uniform random sampling seems to produce very consistent embeddings within a column, shown in Figure 4a. This similarity of result between all three datasets implies that this consistency is a more general observation. Hence, using uniform random sampling to sample data seems to produce good embeddings that are visually representative stand-ins for embeddings of the entire dataset.

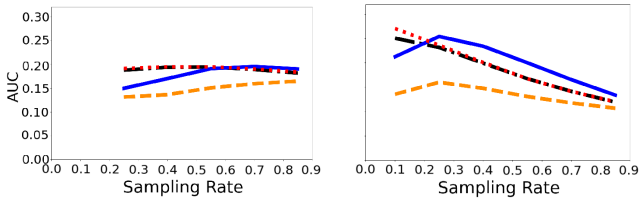
FPS produces somewhat consistent results within a column. As can be seen in Figure 4b, this sampling strategy seems to, once again, give small clusters more weight, causing them to be represented more strongly when compared to uniform random sampling. This is consistent with the findings of the C.Elegans dataset, and could imply that both these datasets don't have a cluster in high-dimensional space that is close to most other clusters, so no significant undersampling can be observed. The representation of otherwise vanishing clusters seems to be a strength of this algorithm that could actually show to be useful for visualisations using this sampling algorithm.

Lastly, PDS seems to once again show consistent results with some splitting of clusters for low sample sizes, as shown in Figure 4c. This shows that using PDS on another dataset produces embeddings with the same properties, making the results found more generally applicable.

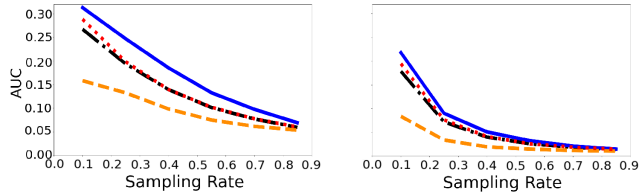
5.2 Precision-Recall evaluation of sampling strategies

Now that we have provided a visual comparison of the different sampling algorithms, we wish to objectify this comparison by performing a numerical analysis. This can be done using the area under the precision-recall curve (AUC) as a measure. Note that this measure produces a value between zero and one, where an area of one would indicate a perfect embedding, and lower values represent worse and worse quality embeddings.

For the MNIST dataset, the effect of sampling rate on the quality of embedding can be seen in Figure 5. In general, there seems to be a trend that higher sampling rates causes a decline in the AUC (see Figures 5b, 5c and 5d). This makes sense because, at higher sampling rates more data points are embedded, and so more data points might contend for the closest neighbour, messing up how well the closest neighbours of the high-dimensional dataset are maintained.



(a) Perplexity of $\frac{1M}{7000}$ (b) Perplexity of $\frac{7M}{7000}$



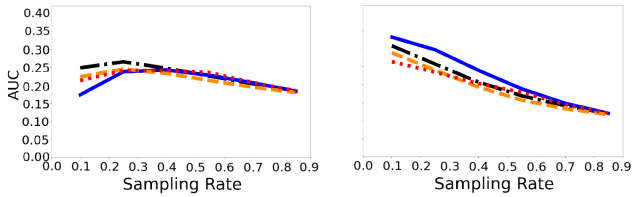
(c) Perplexity of $\frac{21M}{7000}$ (d) Perplexity of $\frac{144M}{7000}$

Figure 5: The average effect (over 4 seeds) of sampling rate on the area under the precision recall curve of the embedding using different sampling strategies on the MNIST dataset. Black (–·) is uniform random sampling, orange (– –) is FPS, blue (–) is PDS, and red (··) is RWS.

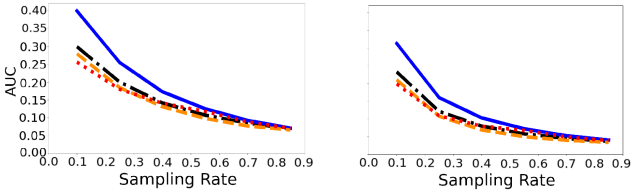
Additionally, Figure 5 reveals that numerically, PDS seems to outperform the other algorithms in most situations. While uniform random sampling sometimes performs better, especially in Figure 5a, FPS is clearly the lowest scoring in all situations. This lines up with the visual analysis, showing that an entire cluster disappears using this sampling strategy. Additionally, it is clear that RWS and uniform random sampling perform the same except when using very small sampling rates, which again lines up with the visual results obtained.

Similar patterns can be observed when the same numerical analysis is run on the C.Elegans dataset, presented in Figure 6. While PDS still appears to perform the best in almost all situations, it is notable that the remaining three sampling strategies

have moved a lot closer together, with no algorithm clearly outperforming or underperforming compared to the others. This is interesting, because it shows that all three algorithms seem to be performing decent, at least when compared numerically in this manner.



(a) Perplexity of $\frac{3M}{8970}$ (b) Perplexity of $\frac{12M}{8970}$



(c) Perplexity of $\frac{34M}{8970}$ (d) Perplexity of $\frac{105M}{8970}$

Figure 6: The average effect (over 4 seeds) of sampling rate on the area under the precision recall curve of the embedding using different sampling strategies on the C.Elegans dataset. Black (−·) is uniform random sampling, orange (−−) is FPS, blue (−) is PDS, and red (··) is RWS.

5.3 Runtime analysis of the embeddings

In addition to the visual and numerical analysis, it is practically relevant to look at the runtime change when using this sampling-based approach. For this, data is collected on how long the embedding of the (sampled) data takes for different sampling rates and perplexity ratios. It is notable that the sampling method used here did not impact the time taken, so the focus is on the timings using uniform random sampling. The results can be seen in Figure 3.

Sample Rate	Perplexity Ratio			
	$\frac{1M}{7000}$	$\frac{7M}{7000}$	$\frac{21M}{7000}$	$\frac{144M}{7000}$
0.1		40.2949	40.0240	59.6335
0.4	55.3980	80.8241	144.3457	762.6433
0.7	75.8105	165.1294	384.0978	2378.9994
1.0	103.8373	307.4926	796.8412	5102.6103

Table 3: The average time (over seeds 12, 22, 32, 42) it takes (in seconds) for the t-SNE embedding to be computed on the MNIST dataset using different sampling rates and perplexity ratios.

This experiment clearly shows that the larger the sample, the longer the runtime. Additionally, higher perplexity values cause the runtime to be much higher as well, especially for larger sampling rates. For these large perplexity values the difference is quite dramatic, going from about 5,103 seconds, or approximately 85 minutes, for embedding the full dataset to about 60 seconds for 10% of the data. Given this large contrast in runtime, a sampling-based approach appears to be very beneficial in comparison to embedding the full dataset.

With this knowledge, it is important to also look at how long the actual sampling of the dataset takes, to ensure that the overall runtime actually decreases. For this, another experiment is run on the MNIST dataset, measuring how long sampling given a certain sample rate takes for each sampling strategy. The results are presented in Table 4. Note that the highest perplexity ratio of $\frac{144M}{7000}$ is used for this experiment, which should only impact the performance of the PDS strategy (see Appendix A).

Sample Rate	Sampling Strategy		
	Uniform	FPS	PDS
0.10	**0.004470**	77.484786	8.120034
0.25	**0.002502**	188.562985	6.389263
0.40	**0.002380**	311.559060	5.729714
0.55	**0.003054**	407.148036	3.922512
0.70	**0.003845**	503.706841	2.958181
0.85	**0.006488**	606.628891	1.716478

Table 4: How long it takes (in seconds) to compute the sample of the MNIST dataset using different sampling strategies. The results presented here are the average of running each sampling algorithm over four different seeds (12, 22, 32, 42). In bold is the best performing sampling strategy for each sample rate.

From the results found in Table 4, it is clear that uniform random sampling takes the least time, and seems to be independent of how large of a sample is being collected. PDS performs quite decent as well, taking longer for smaller samples since more data points have to be eliminated. At the rate of eliminating about 8,000 samples a second, it still scales pretty well. FPS, on the other hand, clearly performs the worst. It seems to sample approximately 90 samples a second, causing larger samples to have significant runtime.

RWS is not included in this experiment since the sample size does not impact the runtime of this sampling algorithm. To sample, 10 walks starting from each data point have to be run once for the entire dataset, and the samples can then be picked from this computation. Selecting the samples from this list takes very little time, but pre-computing all these random walks on the entire dataset is very slow, taking approximately 50 minutes for the MNIST dataset.

Overall, both uniform random sampling and PDS appear relatively fast, meaning users could benefit from the speed-up of embedding a sample of the data. FPS, on the other hand, takes so long to sample that this benefit is removed for most of the perplexity values tested, making it a poor option. Even worse, RWS takes such a significant amount of time to sample that it actually becomes as slow, or slower than simply embedding the entire dataset.

6 Responsible Research

While conducting this research, there were multiple aspects related to responsible research that were important to con-

sider. Most notable were the ethical implications of this study, as well as ensuring the reproducibility of results.

Regarding the ethics of this work, it was important to consider how ethical data collection was during the research process. To address this, well-known and freely available datasets were used, that have all been used in other research regarding a similar field, like in [5]. Doing this ensured unbiased, easily available data was used which did not have any negative ethical implications in its content.

In addition to this, another concern was the potential ethical implications the results of this research might have. The findings of this paper could be misused for data visualisation and research of non-ethical datasets. If used maliciously, this could lead to purposefully biased representations of data. This is a reality that is hard to prevent in this research itself, and so our focus was on ensuring the datasets and conclusions reached in this paper were as ethical and unbiased as possible.

Lastly, we wanted to ensure that our research was reproducible and generalizable. To ensure generalizability, two or, where possible, three datasets were used and the average over four runs was taken for all numerical results. This guaranteed more unbiased results. Additionally, all experiments were conducted using seeds (visualisation: seed 42, numerical: average over seeds 12, 22, 32, 42). By doing this while also making the code used publicly available, we ensured all results in this paper are generalizable and easily reproducible.

Through considering these different aspects, we hope to have shown that the research presented in this paper was conducted ethically, and can be reproduced by anyone who so wishes to.

7 Conclusions and Future Work

In this paper, we presented our research into the impact of using t-SNE to embed a sample of a dataset obtained through different sampling strategies. The aim was to understand the impact of different sampling strategies on both the visual quality of the embeddings, and on the runtime of the overall workflow.

It was found that uniform random sampling performed well, maintaining good quality embeddings that were similar to the full embedding of the dataset even at low sampling rates. Additionally, sampling using this strategy did not take a significant amount of time, even for large samples, meaning it made full use of the decreased runtime achieved by embedding only a part of the dataset. Therefore, uniform random sampling was found to be a good sampling strategy to use in this context.

Farthest point sampling, on the other hand, showed weaker results. Visually, the algorithm did not create consistent images, often varying a lot due to over/undersampling of certain clusters. Numerically, these embeddings were confirmed to be of poor quality, usually maintaining the least neighbours from the high-dimensional dataset. Additionally, the sampling algorithm was slow to run, decreasing the improvements gained by sampling in the first place. Due to this, farthest point sampling was deemed to be a bad sampling strategy to use in this context.

Poisson disk sampling, being the third sampling algorithm that we tested, once again showed promising results. While the embeddings were less visually consistent than for uniform random sampling, this was mostly caused by the appearance of additional clusters. It was shown that this sub-clustering of the sample embedding actually maintained meaningful information about the dataset that was lost in the full embedding. Additionally, the algorithm performed the best numerically in almost all situations, showing that these produced embeddings were good at maintaining the high-dimensional neighbours of the dataset. Lastly, Poisson disk sampling was also relatively fast to run, meaning it actually provided a speed-up. Due to this, this sampling algorithm showed to be a very good option for quickly embedding a sample of the dataset: creating a good embedding that might even maintain more information about the dataset than could be expected in the full dataset embedding.

The last sampling approach that was tested made use of random walks to sample the dataset. These embeddings seemed to both visually and numerically be very similar to uniform random sampling on the MNIST dataset. For the C.Elegans dataset there was some variation, but these were not necessarily shown to be better. Runtime wise, the algorithm was by far the slowest, often taking longer than simply embedding the entire dataset. With these results, this sampling algorithm was also shown to be a poor choice for this use-case.

Hence, we found that both uniform random sampling and Poisson disk sampling were good sampling algorithms to use. Uniform random sampling maintained a lot of consistency in the embedding, meaning an estimation of the visual of the full embedding could be made from a quickly obtained sample. Poisson disk sampling, on the other hand, seemed to maintain high-dimensional neighbourhoods very well, even to the extent that more detailed visualisations were obtained using smaller sample sizes. Which of these two algorithms performed better would depend on the specific application.

While these results were already promising, we suggest for more research into this topic. This would include running the experimentation on other datasets to confirm whether the same trends can be observed. Additionally, we suggest investigating biased sampling strategies, such as one that prioritizes high-density areas in the dataset. This could be done by giving certain areas in the dataset a higher probability of being sampled while using strategies like Poisson disk sampling, random walks, or importance sampling. With more research, the results presented in this paper could be made more extensive, making it more applicable to the high-dimensional data visualisation workflow: allowing individuals to quickly visualise and hence identify patterns in the vast amount of data present on the Internet and in the world, no matter how many dimensions this data might have.

References

[1] IDC and Statista, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028 (in zettabytes) [graph]." https://www-statista-com.tudelft.idm.oclc.org/statistics/871513/worldwide-data-created/, May 2024. Accessed: 05 May 2025.

[2] A. Unwin, "Why Is Data Visualization Important? What Is Important in Data Visualization?," *Harvard Data Science Review*, vol. 2, Jan. 2020.

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

[4] S. Ayesha, M. K. Hanif, and R. Talib, "Overview and comparative study of dimensionality reduction techniques for high dimensional data," *Information Fusion*, vol. 59, p. 44–58, July 2020.

[5] M. Skrodzki, N. F. C. de Plaza, T. Höllt, E. Eisemann, and K. Hildebrandt, "Navigating perplexity: A linear relationship with the data set size in t-sne embeddings," 2024. ArXiv, https://arxiv.org/abs/2308.15513.

[6] M. Wattenberg, F. Viegas, and I. Johnson, "How to use t-sne effectively." https://distill.pub/2016/misread-tsne/, 2016. Accessed: 07 May 2025.

[7] D. Kobak and G. C. Linderman, "Initialization is critical for preserving global data structure in both t-sne and umap," *Nature Biotechnology*, vol. 39, p. 156–157, Feb. 2021.

[8] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xu, and J. Zhu, "QuickFPS: Architecture and Algorithm Co-Design for Farthest Point Sampling in Large-Scale Point Clouds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, pp. 4011–4024, Nov. 2023.

[9] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xie, S. Zheng, and J. Dong, "FuseFPS: Accelerating Farthest Point Sampling with Fusing KD-tree Construction for Point Clouds," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, (Incheon, Korea, Republic of), pp. 238–243, IEEE, Jan. 2024.

[10] J. Li, J. Zhou, Y. Xiong, X. Chen, and C. Chakrabarti, "An Adjustable Farthest Point Sampling Method for Approximately-sorted Point Cloud Data," in *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, (Rennes, France), pp. 1–6, IEEE, Nov. 2022.

[11] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, p. 579–589, June 1991.

[12] T. Wang, "Poisson-Disk Sampling: Theory and Applications," in *Encyclopedia of Computer Graphics and Games* (N. Lee, ed.), pp. 1–8, Cham: Springer International Publishing, 2021.

[13] D. Dunbar and G. Humphreys, "A spatial data structure for fast Poisson-disk sample generation," *ACM Transactions on Graphics*, vol. 25, pp. 503–508, July 2006.

[14] A. Lagae and P. Dutré, "A Comparison of Methods for Generating Poisson Disk Distributions," *Computer Graphics Forum*, vol. 27, pp. 114–129, Mar. 2008.

[15] C. Yuksel, "Sample Elimination for Generating Poisson Disk Sample Sets," *Computer Graphics Forum*, vol. 34, pp. 25–32, May 2015.

[16] R.-H. Li, J. X. Yu, L. Qin, R. Mao, and T. Jin, "On random walk based graph sampling," in *2015 IEEE 31st International Conference on Data Engineering*, (Seoul, South Korea), pp. 927–938, IEEE, Apr. 2015.

[17] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova, "Hierarchical Stochastic Neighbor Embedding," *Computer Graphics Forum*, vol. 35, pp. 21–30, June 2016.

[18] L. van der Maaten, "Barnes-Hut-SNE," Jan. 2013. Version Number: 2, ArXiv, https://arxiv.org/abs/1301.3342.

[19] V. Delchevalerie, A. Mayer, A. Bibal, and B. Frénay, "Accelerating t-sne using fast fourier transforms and the particle-mesh algorithm from physics," in *2021 International Joint Conference on Neural Networks*, p. 1–8, July 2021.

[20] D. Steen, "Precision-recall curves." https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248, Sept. 2020. Accessed: 25 May 2025.

A Impact of Perplexity on Poisson Disk Sampling (PDS) runtime

We chose to implement PDS using the elimination based technique presented in [15]. However, we slightly modified this method to use the distance to a data points' close neighbours as the weights. This meant that the PDS strategy was impacted by the user chosen perplexity value, determining how many close neighbours each data point had.

This choice had an impact on the runtime of the algorithm. The higher the perplexity, the more data points were considered neighbours, and hence the more computations had to be completed. This was numerically confirmed through an experiment, the results of which are shown in Table 5. As can be seen, higher perplexity values resulted in longer runtimes, although the difference was rather small.

Sample Rate	Perplexity Ratio			
	$\frac{1M}{7000}$	$\frac{7M}{7000}$	$\frac{21M}{7000}$	$\frac{144M}{7000}$
0.10		6.034390	6.361625	8.120034
0.25	4.953796	5.090364	5.259932	6.389263
0.40	3.988330	4.030275	4.241249	5.729714
0.55	2.978139	3.076491	3.187566	3.922512
0.70	2.003535	2.117661	2.178661	2.958181
0.85	1.029398	1.041463	1.097814	1.716478

Table 5: How long it took (in seconds) to compute the PDS sample using different perplexity ratios on the MNIST dataset. The results presented here are the average over four different seeds (12, 22, 32, 42).