Reproduction and Extension of Diffusion Recommender Model

Ilayda Bilgin (13519328), Francesco Tinner (14497425), and Luke Chin A Foeng (14603373)

University of Amsterdam

Abstract. Recommender Systems(RS) take into account user interactions when predicting their preferences to present the most relevant data accordingly. Besides the traditional RS methods such as Collaborative Filtering and Content-based Filtering, generative approaches for predicting user interactions have also been recently used. However, due to some downsides like the unstable nature of GANs or limited representation ability of VAEs, a generative method integrating a Diffusion process was proposed. By utilizing the denoising as done in Diffusion models, the novel method DiffRec is able to learn user interactions in a generative manner. To handle the challenges that DiffRec faces with the extensive costs in large scale item prediction and temporal shifts for user actions, L-DiffRec and T-DiffRec were proposed respectively. We extend the study with those models to mainly observe the reproducibility of the previous study, further explore early stopping, effects of clustering in L-DiffRec and we propose a new weighting scheme for T-DiffRec, namely the Learned Temporal Importance Weigthing. The results of the reproduction study proved to be on par with those presented originally, with discrepancies seen only in specific circumstances due to an absence of certain important hyperparameters. In addition to this, improvements came from a higher patience value in early stopping and learnable weights, while not as high or significant improvements were observed with the clustering ablation study.

Keywords: Recommender Systems · Diffusion Models · Generative Models.

1 Introduction

The primary strength of Recommender Systems (RS) is their ability to filter information and present only the most important information to users, ranked by their preferences. The recommendation can take different forms based on the availability of attributes (user-based and item-based) and the way the data is being analyzed. When the recommendations are solely based on historical interactions, and ratings of all users it is called "Collaborative Filtering, while "Content-based Filtering" bases it's recommendations on matching items to the attributes of the user [14] Recent advancements in collaborative filtering allow to take into account, the preferences of other users, and the similarities between items when predicting item-interaction probabilities for a specific user. Ranking items by the predicted interaction probability in decreasing order, allows to generate an ordering of items a user is most interested in, therefore filtering out items that are not relevant to a user. Some of the factors which can influence this task negatively are noisy and sparse data [3].

RS utilize different model architectures as their backbone, which influences the quality of the recommendations produced. Taking advantage of the generative nature of Generative Adversarial Networks (GANs) allowed to address the issues of noisy and sparse data and increase performance on the recommendation task further. This was achieved due to the ability of GANs to more accurately learn the generative process of user interactions. The training of GANs is known to be challenging and unstable, which can lead to unsatisfactory performance due to the failure to converge, or mode collapse [20].

Another architecture that offers a trade-off between tractability and representation ability are Variational Autoencoders (VAEs). Using the observed user interactions VAEs use an encoder to model the posterior distribution and hence maximize the likelihood of the observed interactions. The trade-off results from the reduced ability of a tractable encoder to model diverse user preferences [22].

In the light of recent successes in the domain of image generation, [22] investigate the suitability of a diffusion model as an alternative backbone model for a RS. More details about diffusion models in the context of recommendation can be found in section 2. The present report's focus lies on reproducing and extending the work of [22] and asks the following three research questions:

- **RQ1**: Are the results presented within the paper Diffusion Recommender model [22] reproducible?
- **RQ2**: How does the increased patience metric improve the DiffRec model variations' recommendation performance?
- **RQ3**: How does the addition of Learned Temporal Weights (LTW) improve the recommendation accuracy of the T-DiffRec model?

Ultimately, it was found that the results were largely reproducible when making use of the provided repository and paper, even with the absence of some key hyperparameters. These hyperparameters had to be adjusted during the reproduction process, but the results were comparable to those mentioned within the original paper. In addition to this, the extensions to the paper were fruitful, as they improved the overall results of the original paper by roughly 5% to 10% across the accuracy metrics used within the original paper[1].

2 Related Work

Ambitions to use diffusion-based generative models have been prevalent in recent times. In the following, we summarize and discuss common properties and important differences of recently proposed works [2,9]. The original paper [22] derives its use from the diffusion model, which originally made use of the diffusion process in order to generate a new image through the gradual introduction, and subsequent removal of noise[19,6]. This diffusion process has roots in different types of generative models, mainly GANs [21] and VAE-based models [13]. VAE-based recommender models generally outperform GAN-based systems on the recommendation task, as VAEs are able to capture more nuanced user interaction patterns [22]. However, their complexity also leads to longer training times.
The diffusion-recommender model is able to bridge this by offering a methodology to model complex user interactions, while also keeping the complexity low [17]. Ultimately, existing diffusion architectures designed for image generation appear to have interesting applications in the recommender systems domain, due to the their usage in extracting viable recommendations from noisy data.
Additionally, the inclusion of temporal information in the diffusion process can play an important role in further enhancing recommendations, as each user interaction is associated to a certain timestep, and user preferences are evolving dynamically [11]. This learned timestamped user-interaction methodology has been shown to work independent of diffusion recommender models and has been implemented only as simple fixed linear weighting addition to the DiffRec model via the T-DiffRec sub-method [22]. For more details on the T-DiffRec method, see 3.3.

3 Methodology

3.1 DiffRec

[22] introduce the Diffusion Recommender Model (DiffRec), which utilizes a diffusion model in order to generate recommendations from observed user interactions [19].

Similarly to applying the diffusion process on image generation, it can be applied to interaction data [6]. This ability is valuable, as user interaction data contains noise naturally. In this setting, interaction data is corrupted with random noise and the diffusion model is trained to learn a reconstruction of the original user interactions (x_0) from pure noise (x_T) via Equation 1:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \varSigma_\theta(\mathbf{x}_t, t)). \quad (1)$$

This formula denotes a denoising step which includes the Gaussian parameters μ_θ and $\varSigma_\theta$ relevant to model the distribution of the added noise, which was added to the original interaction inputs at a certain diffusion step t.

[1] Our codebase can be found here: `https://github.com/ilayda-bilgin/RS`

Optimization The optimization of the diffusion process is performed using the a maximization of the evidence lower bound equation (ELBO), where the difference between the reconstruction ($\mathbf{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[logp_\theta(\mathbf{x}_0|\mathbf{x}_1)]$) and denoising ($\Sigma_{t=2q(\mathbf{x}_t|\mathbf{x}_0)}^T[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))]$) equations must be maximized. The equation used for this purpose can be found in Equation 2. The KL divergence is used to approximate the distributions.

$$logp(\mathbf{x}_0) \geq \mathbf{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[logp_\theta(\mathbf{x}_0|\mathbf{x}_1)] - \Sigma_{t=2q(\mathbf{x}_t|\mathbf{x}_0)}^T[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))] \tag{2}$$

The model is trained by increasing the difference between the reconstruction and denoising parameters within the ELBO equation, so as to get as close to $logp(\mathbf{x}_0)$ as possible. For the denoising term, the probability $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is used to approximate the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ through the KL divergence. From here, Bayes rule can be employed to rewrite the term $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$, as a normal distribution $\mathcal{N}(\mathbf{x}_{t-1};\bar{\boldsymbol{\mu}}(\mathbf{x}_t,\mathbf{x}_0,t),\sigma^2(t)\mathbf{I})$. The terms $\bar{\boldsymbol{\mu}}(\mathbf{x}_t,\mathbf{x}_0,t)$ and $\sigma^2(t)\mathbf{I}$ represent the mean and covariance of the previously defined distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$, respectively. During the training of this model, $\Sigma_\theta(\mathbf{x}_t,t)$ from the distribution $p_\theta(\mathbf{x}_{t-1}$, is not leaned but set equal to $\sigma^2(t)\mathbf{I}$. This serves as a stabilization method.

3.2 Latent DiffRec (L-DiffRec)

This extension of DiffRec aims to reduce the costs in terms of computational requirements in large scale-item prediction specifically. The underlying approach and special characteristics of L-DiffRec compared to other generative models are: (1) clustering of the interaction-items, (2) dimensionality reduction using multiple VAEs, referred to as **encoding**, and (3) applying the diffusion process in the latent space, denoted as **decoding** step. An overview of L-DiffRec's architecture can be found in Figure 1, [22].

Encoding In the encoding process, the item representations are clustered into C sets of categories using K-means clustering. Thus, items are grouped together based on their feature similarity. Then, based on an item's cluster assignment, the user interaction vector x_0 is divided into C parts. Compression of each user interaction x_0 vector to z_0 is performed using a variational auto-encoder. With this encoder, the mean μ_{ϕ_c} and the covariance $\sigma_{\phi_c}^2 I$ of the variational distribution $q_{\phi_c}(z_0^c|x_0^c) = N(z_0^c;\mu_{\phi_c}(x_0^c),\sigma_\phi^2(x_0^c)I)$ can be predicted.

Diffusion In contrast to DiffRec, the diffusion process is performed in the latent space. Through concatenation of $\{z_0^c\}_{c=1}^C$ the compressed representation z_0 is obtained on which the forward and reverse diffusion process is performed directly. Optimization loss is also similar to DiffRec's loss : $\mathcal{L}(x_0,\theta) = \mathbb{E}_{t\sim\mathcal{U}(1,\mathcal{T})}\mathcal{L}_t$, the parameters of the denoising multi-layer perception (MLP) are denoted with θ.

Decoding The reconstruction $\hat{z_0}$ resulting from the reverse process is split into $\{\hat{z_0}^c\}_{c=1}^C$ based on the item cluster formed in the step explained in subsection 3.2. Finally, predictions are obtained using a separate decoder, which utilizes a method similar to MultiVAE [10] in order to obtain the predictions $\hat{x_0}$ from $p_{\psi_c}(\hat{x_0}^c|\hat{z_0}^c)$. The goal of the decoding step is to remove false positives and false negatives from the interactions.

3.3 Temporal DiffRec (T-DiffRec)

T-DiffRec is a variation of the original DiffRec model described in subsection 3.1. It assigns to the user interactions, in order to provide additional temporal context to the user actions. Compared to DiffRec, the changes in model architecture are minimal, as only the loss calculation is affected. T-DiffRec weights the importance of actions within a sequence by assigning a larger weight to the loss of more recent interactions. This re-weighting procedure is based on a linear function, the more recent an interaction occurred, the larger the weight. Weights lie on a scale from 0 to 1. This follows the equation $w_m = w_{min} + \frac{m-1}{M-1}(w_{max} - w_{min})$ for each weight w_m in the set of all weights $\mathbf{w}$ per interaction vector.

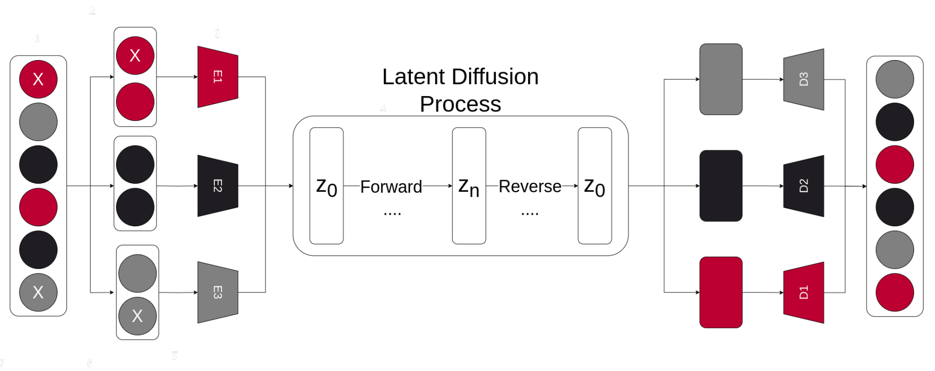


Fig. 1: The L-DiffRec architecture. Input to the diffusion process is the encoded input of x_0. The forward forward pass adds noise repeatedly during n steps. The reverse diffusion process learns the noise, and iteratively removes it, aiming at removing false positives and false negatives (denoted by the $\times$ symbols). Last step is the decoding, where the latent reconstructed representation is converted back to the original space. Own creation, 2023.

3.4 Latent Temporal DiffRec (LT-DiffRec)

This extension of the DiffRec architecture combines the extensions L-DiffRec with T-DiffRec. Thu, it makes use of both, the temporal features, and the diffusion in the latent space. One additional deviation from the L-DiffRec architecture is necessary in order to accommodate for the additional weights the temporal re-weighting pre-processing method creates.

3.5 m-PHATE

The m-PHATE module is a dimensionality reduction method used to better visualize the weights in relation to one another. This visualization methodology is a multi-slice derivative of the PHATE visualization method[2].[15,4]. This method's strenghts lie in its ability to reduce the dimensionality of temporal data by developing a temporally relational graph that connects each weight to itself across all time steps. The connection developed by this multi-slice kernel is then weighted through the use of a similarity score, which is then employed in the creation of a weight relational temporal plot. Interpretation of the evolvement of a tensor during training remains challenging, but helpful insights can be gained by comparing the patterns of the visualizations between methods, features, or datasets.

3.6 Datasets

All the datasets used by [22] contain publicly available reviews on a scale of 1-5 stars. Items to which a certain user gave 4 or more stars, are considered relevant (to the user). This allows the construction of a user interaction vector. Furthermore, each item is characterized by a vector of features, such as genre, duration, year of release - in the case of the movies in the ML-1M dataset. These embedding files were provided by the authors. All in all, three datasets - Yelp, Amazon-book, and ML-1M were used in various experiments. The split into train, validation, and test set follows the rations 7:1:2. It is unclear whether the splitting of the interactions was performed randomly, as neither the details nor the implementation were not provided in the original paper. Calculating the relative differences between the provided item and interaction numbers shows, that the noisy interactions added are different for each dataset. More exactly, the added noise is overproportional for Amazon-book and underproportional for ML-1M. For more details see Table 2.
Each dataset is used under two different settings: noisy and clean. Under the noisy setting noise is added to the training and validation splits, but not to the test set. Noise includes both natural noise from interactions with a rating below the threshold of 4 stars, and randomly sampled interactions.

[2] `https://github.com/scottgigante/m-phate#m-phate`

Differences in terms of user number and item number between the different datasets are significant. (more details can be found in Table 1, and Table 3)

	#Users	**#Items**	**#Interactions**
Amazon-book clean	108822	94949	3146256
Amazon-book noisy	108822	178181	3145223
Yelp clean	54574	34395	1402736
Yelp noisy	54574	77405	1471675
ML-1M clean	5949	2810	571531
ML-1M noisy	5949	3494	618297

Table 1: Dataset statistics

	#Users	**#Items**	**#Interactions**
Amazon-book	0.00%	87.66%	-0.03%
Yelp	0.00%	125.05%	4.91%
ML-1M	0.00%	24.34%	8.18%

Table 2: Relative increase between the clean and the noisy version of the dataset for the number of users, number of items and number of interactions.

	#Users	**#Items**	**#Interactions**
Yelp clean	0.00%	0.00%	0.00%
Yelp noisy	0.00%	125.05%	4.91%
Amazon-book clean	99.40%	176.05%	124.29%
Amazon-book noisy	99.40%	418.04%	124.22%
ML-1M clean	-89.10%	-91.83%	-59.26%
ML-1M noisy	-89.10%	-89.84%	-55.92%

Table 3: Comparison in relative terms for the number of users, number of items, and number of interactions. Percentages calculated are in relation to Yelp clean.

3.7 Reproduction Details

In general, we note two things: (1) The provided code[3] contains many redundancies, as the authors seem to have implemented each variation of DiffRec separately. This makes it hard to spot differences in the methods. (2) The datasets are only provided as already pre-processed *numpy* arrays. It is therefore impossible to reconstruct the precise steps of the authors' preprocessing pipeline. Another note would be the section of the paper explaining data preparation and preprocessing decisions not being clear. We miss concrete details about how the data was split into train, validation, and test sets. Lastly, it remains unclear how the noisy variation of a dataset was constructed, and why a different amount of noise is added to the interactions of different datasets, see Table 2 and Table 3.

Set-up The installation of the provided codebase was mostly straightforward, as the *ReadMe* provided detailed instructions on how to run each method. Additionally, some dependencies were provided there, which made the installation process relatively easy. However, we enhance the set-up and follow common practices by adding a *conda* environment file and instructions to automatically download the additional data and the checkpoints.

Training In some cases, the data paths were hard-coded, but to ensure flexibility we added them as *argparse* arguments. The same holds true for the setting of the random seed, which we also add as an argument. To ensure that the results we get are consistent, we run every model with 5 different seeds

[3] The original code is available here: `https://github.com/YiyanXu/DiffRec`.

and average over the results.
The biggest issue in training we faced was a contradiction in the default training hyperparameters provided in the `main.py` script - for some datasets. This lead to an assertion statement in the `p_sample` function of `gaussian_diffusion.py` which requires the parameter `steps` to be smaller or equal to `self.steps`. `self.steps` which are the diffusion steps that add the noise was set to 100 for DiffRec, 5 for L-DiffRec, 100 for T-DiffRec and 5 for LT-DiffRec. `steps` is the sampling steps specified by `args.sampling_steps` however is set to 0 for DiffRec, 10 for L-DiffRec, 0 for T-DiffRec, and 10 for LT-DiffRec. We were able to resolve these logical contradictions by changing the default values per dateset individually to the hyperparameters indicated in the checkpoints released by [22], as these should contain their best models with optimal hyperparameter settings.
We were confronted with resource issues when running training on the Amazon-book datasets. Loading the training set required overmuch RAM, even after increasing the job's CPU memory limit to 120GB. We therefore changed the number of parallel processes in the dataloader to 0 and change the temporary data matrix in `data_utils.py` from float64 to float32. We reason that this change is not leading to a decreased precision, as the user interaction matrices were binarized by the authors' preprocessing pipeline [22].

Inference Lastly, using checkpoints provided by [22], we were able to perform inference on all datasets but Amazon-book noisy. The checkpoints for that dataset were in an invalid format. For all other datasets and models where checkpoints were provided, we were able to obtain the same results as indicated in the paper.

Hyperparameter Investigation Several hyperparameters were neither mentioned within the paper nor within the codebase. It was initially assumed that the codebase default settings were the properly configured hyperparameters, which was later found to be incorrect. Contact with the initial authors was established regarding this, as some initial results did not reflect those discussed within the original paper. This contact was fruitless, as the authors failed to respond to our queries within the allocated time-frame.

4 Experiments

4.1 Clustering Ablation Study

The L-DiffRec model uses clustering as a part of its architecture, before applying diffusion in the latent space. K-means is the default clustering method used in this extension of DiffRec. As the reason behind this choice of clustering algorithm was not explained in [22], we conduct an ablation study to explore the effects the clustering method and its associated hyperparameters have on recommendation performance. Our experiments involve varying clustering sizes, and further extend the exploration of different clustering methods.

Effects of Cluster Size [22] experiment with different clustering sizes and investigate the effects on evaluation metrics, GPU usage and number of parameters. Results are reported on the Amazon-book dataset under training in the clean setting. In our experiments we apply L-DiffRec on Yelp and ML-1M datasets instead of the Amazon-book dataset, due to our limited resources in terms of compute and time. We use different cluster sizes and both the clean and noisy versions of the datasets to investigate the effect the of the noise. According to the provided checkpoints, `n_cate=2`, the default setting of the cluster size, is 2. Thus, we extend the results by training the model with one cluster (`n_cate=1`, which is the same as performing no clustering) and with 3, 4, 5 clusters respectively. The effects these changes have on recommendation performance in relation to the baseline (using K-means) are explained in subsection 5.1.

Effects of the Clustering Method Different clustering methods are known to show different behaviors in terms of performance, depending on the model as well as the dataset [1]. Therefore, we experiment with two clustering methods other than the default K-means on different datasets.

- **K-means**
  K-means is the default clustering method used in the original model, however the authors do not

provide an explanation for this choice. Possible reasons could be its simplicity and efficiency due to its linear structure and time complexity O(ntk) [7].

– **Gaussian Mixture Model (GMM)**
  We use GMM as the second clustering method. While being similar to K-means in terms of the Expectation Maximization, GMMs can better fit non-circular data compared to K-means as K-means does not take variance into account [18]. Therefore, we reason, this better suited clustering algorithm could lead to an increased performance for recommendation.
– **Hierarchical Agglomerative Clustering (HAC)**
  Since the two other methods are used sequentially, we decide to experiment with HAC, as this hierarchical method utilizes dendrograms obtained from the tree clusters. We investigate the effects this different approach has on recommendation performance. We aim to explore whether this algorithm has an impact on performance. HAC is known to be less sensitive to the noise in the data [7].

4.2 Early Stopping and Patience Value

Early stopping is a regularization method applied to prevent neural network models from overfitting [16]. The original code uses a fixed patience value of 20 for early stopping. Thus, training is stopped early if the recall@20 is not improved over the last 20 epochs. We find that this assumption is not optimal, as the number of epochs needed for convergence varies between datasets and methods. We therefore increase the patience value when training on the smaller datasets to 200 epochs. This feature can be used by adding a `--patience` hyperparameter that has to be set for the run command of each experiment. We reason that the increased training duration is justified on smaller datasets (all datasets used except Amazon-book), as training does not require a lot of compute resources. Furthermore, overfitting is evaded, as we select the best performing model according to the highest recall@20 measured on the validation set. For the results consult subsection 5.1.

4.3 Learned Temporal Importance Weighting

In the previously defined temporal weighting methodology, the model relied on a pre-processing step that involved a linear re-weighting of the interactions based on how recent each interaction was. This method was deemed too rigid, as it may not have necessarily been the case that the temporal weights had to be linear. With this in mind, a new method was conceived, where instead of relying on this linear, re-weighting, a learned temporal weighting method could be implemented instead. This learned weighting method made use of the previous pre-processing method, however, instead of extending the previous interactions with a vector of temporal weights, the pre-processing simply attaches a vector $\mathbf{1}^T$ to the original training dataset. This same process was also applied to the training and validation datasets. The Figure 2 displays the structure of this extension in relation to the original diffusion model, where the vector is initialized as a set of 1 values, signifying a constant initial importance to all timestamps within each set of interactions, with the weighting determining each interaction's significance.

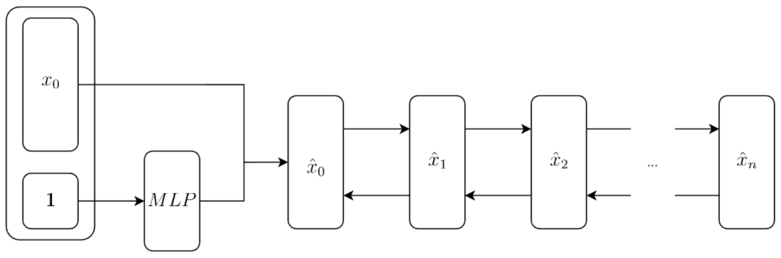


Fig. 2: Learned Temporal DiffRec. Own creation, 2023.

From here, the learnable weights are introduced via a matrix $\mathbf{W}$ of size $m \times n$ where m is the total number of diffusion steps taken within the diffusion process, and n is the total length of the time embedding vector. This gives each diffusion step at each time-step a unique weighting. These are then trained through the time-step importance sampling process conducted prior to the model forward pass within the T-DiffRec and LT-DiffRec flavours of the DiffRec architecture.
This experiment was conducted a total of 5 times for the Temporal DiffRec model with no additional changes to the other hyperparameters. This experiment was only conducted once for the LT-DiffRec model as an exploratory study, but was discontinued due to unsatisfactory results, the suspected reason for which shall be discussed in detail in the subsection 5.1.

5 Evaluation and Results

Overall, the reproduction portion of this paper yielded results that were inline with those presented within the original paper [22]. Each model was ran five times per dataset, except for the Amazon-book dataset. Due to the size of the Amazon-book dataset, it was only run five times with the DiffRec and T-DiffRec models and only with the clean version as it took less time to run with those models. However, for the main reproduction purpose every model was run once, with both clean and noisy versions, to obtain results that are comparable to those of the original paper[22]. The reason for that was to be able to observe the differences that might be caused by the noise and in order to maintain that the results are statistically significant. More runs with the Amazon-book-book datasets would be a potential area of future improvement, as different datasets result in different performances across the models. This future exploration possibility is explained later on as well. The metrics that we report are based on the all-ranking protocol which uses all the items that are not interacted by the users, as mentioned in [5] and also adopted in the Diffusion Recommender System [22].

5.1 Experiment Results

Clustering Ablation Study We conduct a clustering ablation study on the L-DiffRec model which makes use of clustering in its pipeline. All of the experiments are done with `--patience` set to 200 in order to make them more comparable between each other while mitigating the very low results caused by early stopping in different runs. For the first part of our experiments, we use different cluster sizes to observe the effects of clustering. To demonstrate this, we plot K-means with different cluster sizes and datasets. In Figure 3, we can see that there is a slight decrease in performance when the cluster size is increased. Therefore, in this specific study, we note that K-means does not have a positive impact on the performance of the model as it seems to decrease it's performance between no clustering (`n_cate=1`) and more clusters. The results for ML-1M clean with the cluster size 5 is missing as with the clustering in K-means, as there were not enough datapoints in this dataset to obtain 5 distinct clusters.

For the second part, with the default clustering size `n_cate=2`, we report the divergence in percentage between K-means and the other clustering methods, GMM and HAC respectively as can be seen in 4. We can see that the results are highly dependent on the dataset, as we can observe improvements up to 5.18 % in the Yelp clean dataset, regardless of the clustering method, whereas this is not the case with other datasets. With the Ml-1M clean dataset, only HAC has some improvements that are lower than 1 %, while GMM seems to perform better than HAC on ML-1M noisy dataset, even though there is still a decrease in performance compared to K-means. As also mentioned in the comparative study for clustering methods [7], HAC is less sensitive to noise in the data compared to K-means, however this can only be observed on the Yelp noisy dataset in our experiments. Therefore, we can conclude by saying that the performance on the model varies depending on the dataset or the clustering method, as there are some improvements compared to the default clustering method K-means. In Figure 4 the divergence between the default clustering method K-means and Gaussian Mixture Model (GMM), Hierarchical Agglomerative Clustering (HAC) can be seen, with the default clustering (`n_cate=2`), and patience 200.

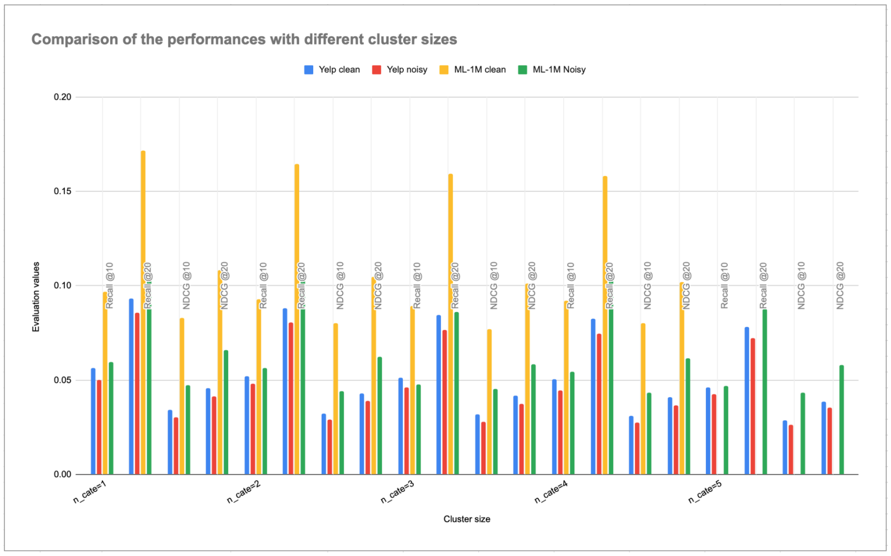


Fig. 3: The comparison of different cluster sizes and datasets with K-means clustering, patience=200. The results are obtained with 4 different datasets in total: Yelp and Ml-1M datasets, under the clean and noisy conditions. The evaluation scores are given with Recall@10, Recall@20, NDCG@10 and NDCG@20.

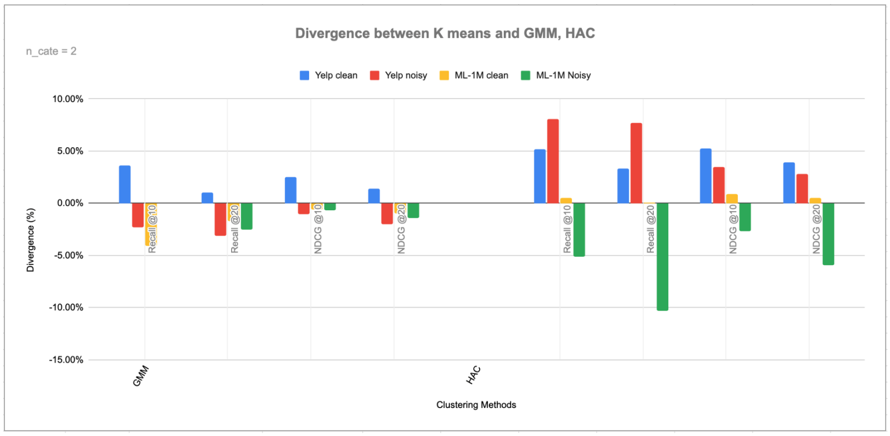


Fig. 4: The difference between K-means, Gaussian Mixture Model (GMM), and Hierarchical Agglomerative Clustering (HAC) clustering methods. Divergence is shown in percentage points.

Early Stopping and Patience Value The results on all metrics show an increase in performance when the increased patience value is applied, suggesting that the model did not converge using the values provided by the original work's authors. A thorough investigation on *ml-1m_clean* showed an average increase in all metrics of 32.9%. The biggest increase in performance occurred on recall@10 with a plus of 65.71% (from 5.6% to 9.3%).

Learned Temporal Importance Weighting Ultimately, the use of learnable temporal weights proved to be a successful extension, yielding improved results when compared to the results of the original implementation. The noisy results of this experiment can be seen in Table 4, where individual metrics were improved by up to 7.7%. This extension proved to give a larger improvement on the noisy datasets than on the clean ones, specifically for the yelp dataset. As for the clean datasets, the results can be seen in Table 5, where there was no noticeable improvement or reduction in any of the metrics when tested on the Yelp dataset. This is in stark contrast to the results presented for the ML-1M dataset, where a significant jump in the metrics can be observed.

	Yelp Noisy				ML-1M Noisy				Amazon-book Noisy			
Model	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
T-DiffRec	0.0426	0.0701	0.0260	0.0345	0.0726	0.1310	0.0549	0.0772	0.0356	0.0499	0.024	0.0284
T-DiffRec + LTW	0.0453	0.0741	0.0280	0.0368	0.0770	0.1398	0.0582	0.0818	0.0356	0.0499	0.024	0.0284
% Improvement	6.3 %	5.7%	7.7%	6.7%	6.1%	6.7%	5.6%	6.0%	0	0	0	0

Table 4: Noisy Results of the Learnable Temporal Weights Extension.

	Yelp Clean				ML-1M Clean				Amazon-book Clean			
Model	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
T-DiffRec	0.0524	0.085	0.0329	0.0428	0.1041	0.1771	0.0932	0.1186	0.06	0.0835	0.0422	0.0493
T-DiffRec + LTW	0.0527	0.0848	0.0330	0.0428	0.1131	0.1885	0.09906	0.1255	0.06	0.0835	0.0422	0.0493
% Improvement	0.5 %	-0.2%	0.3%	0%	8.6%	6.4%	6.3%	5.8%	0	0	0	0

Table 5: Clean Results of the Learnable Temporal Weights Extension.

The Amazon dataset presented several difficulties due to its size. This ultimately resulted in issues with running experiments using it. Specifically, when running the LTW experiments on the clean and noisy Amazon datasets, the results came out to be the exact same as the normal implementation. This is suspected to be due to an anomaly in the setup, even though the same setup was used to produced all the results for the LTW experiments. The results for the LT-DiffRec model with LTW were less than optimal when trained on all datasets, barring the Amazon clean and dirty datasets due to their size. The results for the clean datasets can be seen in Table 6, while the results for the noisy datasets can be seen in Table 7.

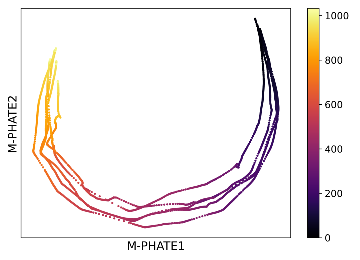
	Yelp Clean				ML-1M Clean			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
LT-DiffRec	0.0589	0.0967	0.0364	0.0478	0.0968	0.1654	0.084	0.1084
LT-DiffRec +LTW	0.0583	0.0967	0.0359	0.0475	0.0895	0.151	0.0833	0.1047
% Improvement	-1.02%	0.00%	-1.37%	-0.63%	-7.54%	-8.71%	-0.83%	-3.41%

Table 6: LT-DiffRec Clean Datasets

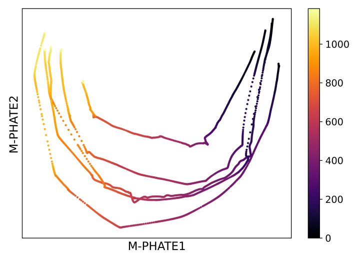
	Yelp Noisy				ML-1M Noisy			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
LT-DiffRec	0.0514	0.0868	0.0314	0.0422	0.0585	0.114	0.0445	0.0656
LT-DiffRec +LTW	0.0363	0.0612	0.0252	0.0332	0.0593	0.1065	0.0478	0.0659
% Improvement	-29.38%	-29.49%	-19.75%	-21.33%	1.37%	-6.58%	7.42%	0.46%

Table 7: LT-DiffRec Noisy Datasets

Visualization of the Learned Weighting In addition to the learned temporal weighting, some mPHATE plots were constructed. These plots help to associate the learned weights with one another, as well as how they developed over time to their final values. Since these values are relational, and are ultimately clustered versions of each embedding vector, their relation is only shown visually, and each axis shown spans a linear scale from 0 to 1. In Figure 5, the evolution of these learnable parameters is presented. It is worth noting that each line in the figures represents a single set of embedding weights per diffusion step. It can also be noted that the the two datasets seem to display comparable mPHATE charts between their clean and noisy versions. In Figure 6, the LTW values can be seen when trained on the ML-1M dataset with a patience of 200.

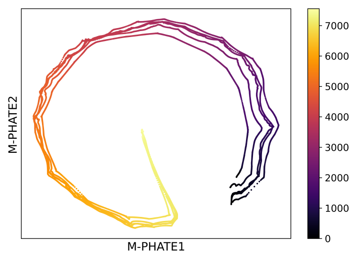


(a) ML-1M Clean Learned Parameters

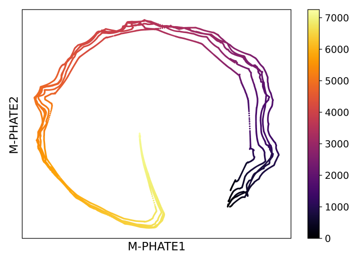


(b) ML-1M Noisy Learned Parameters

Fig. 5: Comparison of T-DiffRec ML-1M Clean and Noisy learned parameters.



(a) ML-1M Clean Extended Patience Learned Parameters



(b) ML-1M Noisy Extended Patience Learned Parameters

Fig. 6: Comparison of T-DiffRec ML-1M Clean and Noisy learned parameters with extended patience.

Resource Investigation After running these experiments, it is also worth noting several peripheral statistics that also play a role in the overall feasibility of each model reproduced within this work. These statistics include model runtime (Figure 7, as well as carbon emissions due to GPU employment.

CO2 Emission Related to Experiments Experiments were conducted using the LISA compute cluster. Although no official information is provided regarding the power source, we reason that it most closely resembles the AWS region eu-west-2, which has a carbon efficiency of 0.62 kgCO_2eq/kWh. A cumulative of 1000 hours of computation was performed on hardware of type Titan RTX (TDP of 280W). Total emissions are estimated to be 173.6 kgCO_2eq of which 0 percents were directly offset by the cloud provider. This amounts to 87 kgs of coal burned. Estimations were conducted using the MachineLearning Impact calculator [4] presented in [8].

[4] https://mlco2.github.io/impact#compute

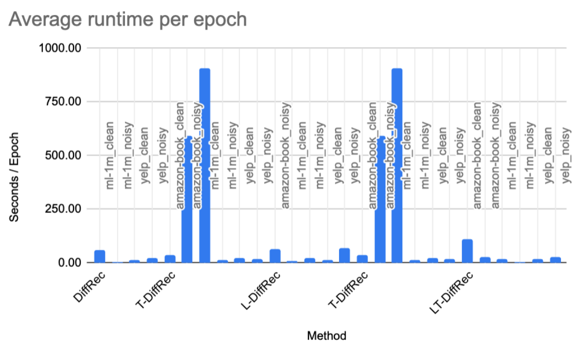


Fig. 7: Average runtimes over methods and datasets in seconds per epoch.

6 Conclusion and Future Work

In this study, we investigated the DiffRec model [22] and its extensions thoroughly. We were confronted with several challenges: In some cases, not enough information was provided explaining the hyperparameter settings. Secondly, logical inconsistencies in the code needed to be solved that occurred with the default hyperparameter settings. Thirdly, an invalid checkpoint was provided. Nevertheless, we could still infer sensible hyperparameter settings, and could train and reproduce all the extensions of DiffRec to some extent as explained in subsection 3.7. Our extensions include a clustering ablation study, using the L-DiffRec model. This revolved around trying different cluster sizes, methods and datasets in which we did not observe significant improvements. However, we could conclude that performance depends on the datasets as well as the methods. Furthermore, we discovered a non-optimal setting regarding the early stopping criteria under which certain models drastically underfitted. We note the increase in performance with a higher patience value in our experiments.

Our main extension was the implementation of Learned Temporal Weighting. As we hypothesized, this enhancement resulted in significant improvements.
For future work, we believe it would be of great value to explore the pre-processing in detail, also it remains open as to why these scripts were not provided by the authors. Another interesting direction might be to experiment with other datasets and conducting more experiments on larger datasets such as the Amazon-book dataset. We were not able to do this due to time and resource constraints.

Lastly, we experimented with additional modifications to the temporal weighting, utilizing firstly a naive transformer-based approach, trying to apply the attention mechanism for sequences. This was unsuccessful due to the large length of user interactions which required more resources than were available to us. A second approach in this direction was also undertaken following the STAMP process[12]. A first implementation and integration into the T-DiffRec model was successful and is provided in our codebase, however further work is necessary to ensure the proper functioning of the implementation.

References

[1] Ahmed, S.R.A., Al Barazanchi, I., Jaaz, Z.A., Abdulshaheed, H.R.: Clustering algorithms subjected to k-mean and gaussian mixture model on multidimensional data set. Periodicals of Engineering and Natural Sciences **7**(2), 448–457 (2019)

[2] Du, H., Yuan, H., Huang, Z., Zhao, P., Zhou, X.: Sequential recommendation with diffusion models (2023)

[3] Gao, M., Zhang, J., Yu, J., Li, J., Wen, J., Xiong, Q.: Recommender systems based on generative adversarial networks: A problem-driven perspective. Information Sciences **546**, 1166–1185 (2021)

[4] Gigante, S., Charles, A.S., Krishnaswamy, S., Mishne, G.: Visualizing the phate of neural networks. Advances in neural information processing systems **32** (2019)

[5] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., Wang, M.: Lightgcn: Simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. pp. 639–648 (2020)

[6] Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems **33**, 6840–6851 (2020)

[7] Kaushik, M., Mathur, B.: Comparative study of k-means and hierarchical clustering techniques. International Journal of Software & Hardware Research in Engineering **2**(6), 93–98 (2014)

[8] Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T.: Quantifying the carbon emissions of machine learning. arXiv preprint arXiv:1910.09700 (2019)

[9] Li, Z., Sun, A., Li, C.: Diffurec: A diffusion model for sequential recommendation (2023)

[10] Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. In: Proceedings of the 2018 world wide web conference. pp. 689–698 (2018)

[11] Liu, K., Shi, X., Kumar, A., Zhu, L., Natarajan, P.: Temporal learning and sequence modeling for a job recommender system. In: Proceedings of the Recommender Systems Challenge, pp. 1–4 (2016)

[12] Liu, Q., Zeng, Y., Mokhosi, R., Zhang, H.: Stamp: Short-term attention/memory priority model for session-based recommendation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. p. 1831–1839. KDD '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3219950, `https://doi.org/10.1145/3219819.3219950`

[13] Ma, J., Zhou, C., Cui, P., Yang, H., Zhu, W.: Learning disentangled representations for recommendation. Advances in neural information processing systems **32** (2019)

[14] Melville, P., Sindhwani, V.: Recommender systems. Encyclopedia of machine learning **1**, 829–838 (2010)

[15] Moon, K.R., van Dijk, D., Wang, Z., Gigante, S., Burkhardt, D.B., Chen, W.S., Yim, K., Elzen, A.v.d., Hirn, M.J., Coifman, R.R., et al.: Visualizing structure and transitions in high-dimensional biological data. Nature biotechnology **37**(12), 1482–1492 (2019)

[16] Prechelt, L.: Early Stopping - But When?, pp. 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg (1998). https://doi.org/10.1007/3-540-49430-8_3, `https://doi.org/10.1007/3-540-49430-8_3`

[17] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10684–10695 (2022)

[18] Shakoor, D.M., Maihami, V., Maihami, R.: A machine learning recommender system based on collaborative filtering using gaussian mixture model clustering. Mathematical Methods in the Applied Sciences (2021)

[19] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International Conference on Machine Learning. pp. 2256–2265. PMLR (2015)

[20] Thanh-Tung, H., Tran, T., Venkatesh, S.: On catastrophic forgetting and mode collapse in generative adversarial networks. CoRR **abs/1807.04015** (2018), `http://arxiv.org/abs/1807.04015`

[21] Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., Zhang, D.: Irgan: A minimax game for unifying generative and discriminative information retrieval models. In: Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval. pp. 515–524 (2017)

[22] Wang, W., Xu, Y., Feng, F., Lin, X., He, X., Chua, T.S.: Diffusion recommender model. arXiv preprint arXiv:2304.04971 (2023)

Appendix

A Individual Contributions

	Extension concept	Implementation	Report	Poster
Ilayda Bilgin	33.3%	33.3%	33.3%	33.3%
Francesco Tinner	33.3%	33.3%	33.3%	33.3%
Luke Chin A Foeng	33.3%	33.3%	33.3%	33.3%
Total	100%	100%	100%	100%

Table 8: Individual contributions.

From the beginning of the project, we all started with brainstorming for extension ideas. We all communicated within the group every possible idea and how it would make sense to split them up. After having reproduced main results from the already existing versions of the models, we moved on to the extensions. While Ilayda was focusing on the clustering ablation study with the L-DiffRec model, Luke was working on the experiments with T-DiffRec and LT-DiffRec in order to have learnable weight parameters, and Francesco was working on that and another possible extension which was a different attention mechanism for T-DiffRec. We worked together on the report and the poster for the final step of our project.

B Reproduction Results Tables

In this section the detailed reproduction results can be found. On most datasets and methods the average results of runs with 5 different seeds are presented. See Table 9 and Table 11. For LT-DiffRec, only the results on the smaller datasets are reproduced due to the heavy computational requirements (Table 12). The same reasoning applies to L-DiffRec on the Amazon-book datasets, were the results obtained are based on one run only (Table 10).

	Yelp				**ML-1M**				**Amazon-book**			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
Clean	0.05468	0.08896	0.034	0.04444	0.09452	0.16252	0.08604	0.10856	0.0588	0.0849	0.0385	0.0464
Noisy	0.04646	0.07718	0.02846	0.03786	0.06326	0.12086	0.04698	0.06856	0.0438	0.065	0.0269	0.0334

Table 9: Average performance over runs using 5 different seeds with DiffRec.

	Yelp				**ML-1M**				**Amazon-book***			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
Clean	0.04146	0.06968	0.02558	0.03424	0.05346	0.10016	0.05976	0.07362	0.0037	0.0069	0.0024	0.0034
Noisy	0.04386	0.07378	0.0267	0.0358	0.0461	0.08966	0.04434	0.05946	0.0035	0.0069	0.0023	0.0033

Table 10: Average performance over runs using 5 different seeds with L-DiffRec
* It was only ran once with the Amazon-book datasets

	Yelp				**ML-1M**				**Amazon-book**			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
Clean	0.0525	0.08466	0.03294	0.0427	0.11224	0.18814	0.0983	0.12492	0.06144	0.08564	0.04298	0.05026
Noisy	0.04554	0.07454	0.02816	0.03702	0.07506	0.13606	0.0573	0.08016	0.03266	0.04748	0.02164	0.02618

Table 11: Average performance over runs using 5 different seeds with T-DiffRec

	Yelp				**ML-1M**			
	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑	R@10 ↑	R@20 ↑	N@10 ↑	N@20 ↑
Clean	0.0583	0.09642	0.036	0.04758	0.09558	0.16224	0.0829	0.10648
Noisy	0.05208	0.08742	0.03164	0.04238	0.05992	0.11266	0.04588	0.06582

Table 12: Average performance over runs using 5 different seeds with LT-DiffRec

C Runtime Evaluation

ntimes for ama-
n missing on
frec (proba-
y done before
ndb

Method	**Data**	**Sec. / Epoch ↓**
DiffRec	ml-1m_clean	61.76
DiffRec	ml-1m_noisy	2.20
DiffRec	yelp_clean	12.25
DiffRec	yelp_noisy	24.80
T-DiffRec	amazon-book_clean	34.93
T-DiffRec	amazon-book_noisy	594.41
T-DiffRec	ml-1m_clean	906.00
T-DiffRec	ml-1m_noisy	13.62
T-DiffRec	yelp_clean	20.63
T-DiffRec	yelp_noisy	18.72
L-DiffRec	amazon-book_noisy	66.60
L-DiffRec	ml-1m_clean	10.96
L-DiffRec	ml-1m_noisy	20.68
L-DiffRec	yelp_clean	13.15
L-DiffRec	yelp_noisy	69.59
T-DiffRec	amazon-book_clean	34.93
T-DiffRec	amazon-book_noisy	594.41
T-DiffRec	ml-1m_clean	906.00
T-DiffRec	ml-1m_noisy	13.62
T-DiffRec	yelp_clean	20.63
T-DiffRec	yelp_noisy	18.72
LT-DiffRec	amazon-book_clean	110.33
LT-DiffRec	amazon-book_noisy	29.51
LT-DiffRec	ml-1m_clean	17.33
LT-DiffRec	ml-1m_noisy	2.06
LT-DiffRec	yelp_clean	19.40
LT-DiffRec	yelp_noisy	27.56

Table 13: Average runtime in seconds per epoch. Runs are only considered with default parameters

D All Reproduction Results

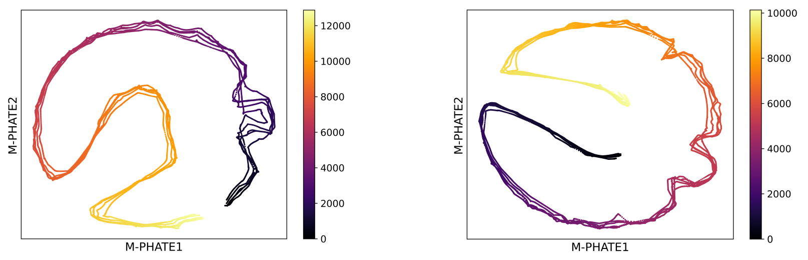
	Yelp Clean				ML-1M Clean				Amazon Clean			
	R@10 ↑	**R@20** ↑	**N@10** ↑	**N@20** ↑	**R@10** ↑	**R@20** ↑	**N@10** ↑	**N@20**	**R@10** ↑	**R@20**	**N@10**	**N@20** ↑
DiffRec	0.0581	0.0960	0.0363	0.0478	0.1058	0.1787	0.0901	0.1148	0.0695	0.1010	0.0451	0.0547
L-Diffrec	0.0585	0.0970	0.0353	0.0469	0.1060	0.1809	0.0868	0.1122	0.0694	0.1028	0.0440	0.0540
T-DiffRec	0.0601	0.0987	0.0377	0.0494	NA	NA	NA	NA	0.0819	0.1139	0.0565	0.0661
LT-DiffRec	0.0604	0.0982	0.0369	0.0484	NA	NA	NA	NA	0.0838	0.1188	0.0560	0.0665
DiffRec Reprod	0.05468	0.08896	0.034	0.04444	0.09452	0.16252	0.08604	0.10856	0.0588	0.0849	0.0385	0.0464
L-Diffrec Reprod	0.04146	0.06968	0.02558	0.03424	0.05346	0.10016	0.05976	0.07362	0.0037	0.0069	0.0024	0.0034
T-DiffRec Reprod	0.0525	0.08466	0.03294	0.0427	0.11224	0.18814	0.0983	0.12492	0.06144	0.08564	0.04298	0.05026
LT-DiffRec Reprod	0.0583	0.09642	0.036	0.04758	0.09558	0.16224	0.0829	0.10648	0.0738	0.1069	0.0476	0.0576

Table 14: Clean results from our reproduction, NA values were not reported by [22].

	Yelp Noisy				ML-1M Noisy				Amazon Noisy			
	R@10 ↑	**R@20** ↑	**N@10** ↑	**N@20** ↑	**R@10** ↑	**R@20** ↑	**N@10** ↑	**N@20** ↑	**R@10** ↑	**R@20** ↑	**N@10**	**N@20** ↑
DiffRec	0.0507	0.0853	0.0309	0.0414	0.0658	0.1236	0.0488	0.0703	0.0546	0.0822	0.0335	0.0419
L-Diffrec	0.0521	0.0876	0.0311	0.0419	0.0665	0.1272	0.0493	0.071	0.0586	0.0876	0.0347	0.0434
T-DiffRec	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
LT-DiffRec	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
DiffRec Reprod	0.04646	0.07718	0.02846	0.03786	0.06326	0.12086	0.04698	0.06856	0.0438	0.065	0.0269	0.0334
L-Diffrec Reprod	0.04386	0.07378	0.0267	0.0358	0.0461	0.08966	0.04434	0.05946	0.0035	0.0069	0.0023	0.0033
T-DiffRec Reprod	0.04554	0.07454	0.02816	0.03702	0.07506	0.13606	0.0573	0.08016	0.03266	0.04748	0.02164	0.02618
LT-DiffRec Reprod	0.05208	0.08742	0.03164	0.04238	0.05992	0.11266	0.04588	0.06582	0.0592	0.0892	0.0359	0.045

Table 15: Noisy results from our reproduction, NA values were not reported by [22].

E Additional m-Phate Visualizations



(a) Yelp Clean Learned Parameters (b) Yelp Noisy Learned Parameters

Fig. 8: Comparison of T-DiffRec Yelp Clean and Noisy learned parameters with extended patience.