A. DATASET PRE-PROCESSING AND DESCRIPTION

To evaluate the performance of DC-Rec model, we conduct experiments on two real-world datasets, i.e., YooChoose1/64[2] and NowPlaying[3]. We conduct the same dataset processing procedure on these datasets following [6, 10]. Specifically, we filter out sessions with length of 1 and items that appear less than 5 times over all sessions. Moreover, we divide the dataset according to the chronological order—we set sessions of the last week/day as the test set, and set the remaining data as the training set. We serially generate item sequence data and corresponding target item labels via a session splitting data augmentation process. For the input session sequence $s = v_1^s, v_2^s, \cdots, v_l^s$, we generate $l-1$ (data, label) tuplets, i.e., $([v_1^s], v_2^s), ([v_1^s, v_2^s], v_3^s), \cdots, ([v_1^s, v_2^s, \cdots, v_{l-1}^s], v_l^s)$. The statistics of datasets after processing is shown in Table 3.

Table 3: Dataset Statistics.

Dataset	YooChoose	NowPlaying
# of training sessions	$369,859$	$825,304$
# of test sessions	$55,898$	$89,824$
# of items	$16,766$	$60,417$
# of clicks	$557,248$	$1,367,963$
Average length	6.16	7.42

B. PARAMETER SETTINGS

We implement our model by PyTorch and release our codes publicly. We search the optimal embedding dimension d in the range of $\{32, 64, 128, 256\}$, and in most cases setting $d = 128$ leads to the best performance. The whole model training can be divided into pre-training process and fine-tuning process. In the pre-training process, we separately train the single channel model and initialize the corresponding embedding vectors with normal distribution with mean of 0 and standard deviation of 0.1. In the fine-turning process, we initialize embedding vectors in DC-Rec with the dumped embedding values. We set the learning rate as $1e-3$ and batch size as 300. We use the Adam as the optimizer. For graph pre-processing, we set the value of N for Top-N sampling in global graph construction as 12. We search the optimal layer depth L in the **GVRL** module in the range of $\{1, \cdots, 4\}$. We set $Recall@20$ as the evaluation metric on the validation set, and early-stop the training process for 3 epochs if the evaluation metric stops improving.

We implement and fine-tune baseline models based on either their official codes or RecBole framework [22]. To make a fair comparison, we set the embedding dimensions of all baseline models as 128. Furthermore, to accelerate the convergence and ensure the baseline performance while making a fair comparison, we set the learning rate as $1e-3$ and set the value of patience steps in early-stop as 5. We fine-tune all baseline models, and search for the optimal parameter assignment. Specifically, regarding SR-GNN, we opt for BPR loss as the loss term, employ a one-layer GRU module, and assign the dropout ratio of the model as 0.5. For NARM, we set the batch size as 300 for fine-tuning. For other hyper-parameters of all baselines, we adhere to their optimized setting values in their papers.

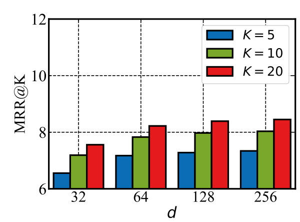
[2] http://2015.recsyschallenge.com/challege.html

[3] http://dbis-nowplaying.uibk.ac.at/#nowplaying

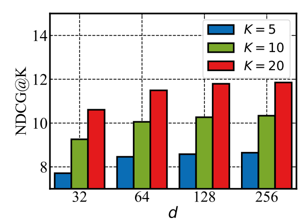
C. MORE DETAILED PARAMETER ANALYSES

We conduct analysis experiments on the key hyper-parameters of DC-Rec. Besides SSL loss ratio β, we also analyze hyper-parameters including embedding dimension d and graph neural network layers L.

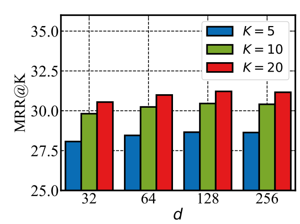
C.1. Impact of Embedding Dimension d



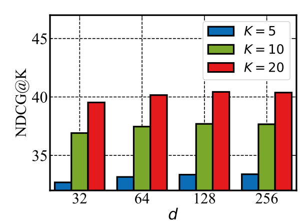
(a) $Recall@K$ on NowPlaying



(b) $NDCG@K$ on NowPlaying



(c) $Recall@K$ on YooChoose



(d) $NDCG@K$ on YooChoose

Fig. 3: Effect of Dimension d

We uniformly set the dimension of embeddings in DC-Rec as d to reduce computation complexity. To investigate the influence of different embedding size, we search for the optimal value of embedding dimension d between $32, 64, 128, 256$. Fig. 3 exhibits the performance comparison of different embedding dimension d. We have the following observations: 1) DC-Rec performs better with the increase of embedding dimension d. 2) DC-Rec tends to perform steady after $d > 128$. 3) We set $d = 128$ uniformly to balance the model performance and storage capacity.

C.2. Impact of Layer L

Table 4: Impact of GNN Layer Depth L

Dataset	L	M@5	M@10	M@20	N@5	N@10	N@20
YooChoose	L=1	28.63	30.41	31.17	33.35	37.67	40.37
	L=2	28.66	30.46	31.22	33.35	37.70	40.43
	L=3	28.62	30.42	31.18	33.31	37.66	40.39
NowPlaying	L=1	7.31	8.00	8.41	8.61	10.28	11.80
	L=2	7.28	7.97	8.39	8.58	10.27	11.79
	L=3	7.26	7.97	8.39	8.53	10.26	11.79

We also conduct comparative experiments on DC-Rec with different convolution layers. Graph channel performance can benefit from stacking multiple convolution layers since capturing wider scopes and model node embedding propagation and aggregation through multi-hops provides the model more powerful expressiveness.

Towards better graph channel performance, we denote DC-Rec with different graph convolution depth L as DC-Rec-L and conduct experiments with representative layer depths, i.e., $L = \{1, 2, 3\}$. Table 4 exhibits the performance of different layer depths (we abbreviate $MRR@K$ as $M@K$ and $NDCG@K$ as $N@K$ for simplicity). As can be seen in the table, on YooChoose, $L = 2$ leads to a consistent outperformance compared with other variants, and on NowPlaying, $L = 1$ leads to the best performance. Jointly analyzing Table 4 with Table 1, we can see that DC-Rec-L consistently outperforms the state-of-the-art methods under every circumstances.