

APPENDIX A: DIFFERENTIAL PRIVACY AND PROOF FOR THEOREM 1

Differential privacy is a privacy mechanism that introduces randomness to prevent information leakage of individuals in datasets (Dwork et al., 2006).

Definition 3 (Differential Privacy) *Formally, a randomized mechanism $\mathcal{M} : \mathcal{X} \rightarrow \mathcal{Y}$ satisfies ϵ -DP if for all neighboring inputs $X \sim X'$ and for all sets of outputs $Y \subseteq \mathcal{Y}$, the following inequality holds:*

$$P[\mathcal{M}(X) \in Y] \leq e^\epsilon P[\mathcal{M}(X') \in Y]. \quad (12)$$

In the above definition, $\epsilon > 0$ is the privacy parameter quantifying the privacy guarantees. Specifically, the smaller ϵ is, the tighter the bound is, indicating less change is allowed in the output distribution when modifying X to X' . Furthermore, if the modification from X to X' is too large, leading to a large change in P , then ϵ may also be large to make \mathcal{M} ϵ -DP. In this case, the privacy is less protected.

From a high-level point of view, we regard each user profile in our recommendation problem as a database in the definition of DP. Consequently, each interacted item within a profile can be regarded as a record in a database. According to the definition of DP, if a mechanism satisfies DP, an observer analyzing its output cannot tell whether a particular data record was used in the computation. As such, applying DP to user profile generation, the recommender functions as the observer. By ensuring DP, the recommender cannot tell whether a particular item from the user's history was used to compute the item scores. Therefore, when we formulate the above process into our user profile generation, it is straightforward that: with limited perturbations, the generated user profile will not cause a significant change in the resulting scores if the entire generation satisfies DP. Finally, the above process can also be prescribed by the mathematical definition of DP: the output distribution of a DP mechanism will exhibit a limited change, if the input is perturbed with a limited budget.

In this appendix, we provide proof for Theorem 1, which is adapted from Wang et al. (2021).

Theorem 1 (Preference-Preservation with Guarantees) *Suppose a recommendation mechanism \mathcal{M} satisfies ϵ -DP. For any user sequence x , if \mathcal{M} successfully recommends the ground-truth item y , which is the j -th item within the item scope \mathcal{I} : $y := x_j$, and the predictive score for the ground-truth item is greater than the second-largest runner-up score of another item with a small multiplicative factor $e^{2\epsilon}$:*

$$\mathbb{E}(\mathcal{M}(x)_j) > e^{2\epsilon} \max_{k:k \neq j} \mathbb{E}(\mathcal{M}(x)_k),$$

then for its augmented view x' , its predicted top-1 item remains the same if there is only limited modification from x to x' :

$$\mathbb{E}(\mathcal{M}(x')_j) > \max_{k:k \neq j} \mathbb{E}(\mathcal{M}(x')_k).$$

Proof 1

$$\begin{aligned} \mathbb{E}(\mathcal{M}(x)_j) &= \int_0^1 \mathbb{P}(\mathcal{M}(x)_j > t) dt \\ &\leq \int_0^1 \mathbb{P}(e^\epsilon \mathcal{M}(x')_j > t) dt \\ &= e^\epsilon \int_0^1 \mathbb{P}(\mathcal{M}(x')_j > t) dt \\ &= e^\epsilon \mathbb{E}(\mathcal{M}(x')_j). \end{aligned} \quad (13)$$

Therefore, we have

$$\begin{aligned} \mathbb{E}(\mathcal{M}(x)_j) &\leq e^\epsilon \mathbb{E}(\mathcal{M}(x')_j) \\ \mathbb{E}(\mathcal{M}(x')_k) &\leq e^\epsilon \mathbb{E}(\mathcal{M}(x)_k), \quad k \neq j \end{aligned} \quad (14)$$

Finally, we have

$$\begin{aligned}
\mathbb{E}(\mathcal{M}(x')_j) &\geq \frac{\mathbb{E}(\mathcal{M}(x)_j)}{e^\epsilon} \\
&\geq \frac{e^{2\epsilon} \max_{k:k \neq j} \mathbb{E}(\mathcal{M}(x)_k)}{e^\epsilon} \\
&= e^\epsilon \max_{k:k \neq j} \mathbb{E}(\mathcal{M}(x)_k) \\
&\geq \max_{k:k \neq j} \mathbb{E}(\mathcal{M}(x')_k).
\end{aligned} \tag{15}$$

Finally, our approach is closely related to the findings in the cited (Wang et al., 2021; Lecuyer et al., 2019). In (Wang et al., 2021; Lecuyer et al., 2019), DP is used to provide certified robustness of classifiers against adversarial examples. That is, in these studies, the goal is to ensure that a classifier’s predictions remain consistent even if the input is deliberately perturbed. Inspired by [1,2], our method leverages DP to generate augmented user profiles that preserve user preferences. In the context of recommendation, this means that the model should produce consistent recommendations for the original user profile and the augmented one if they represent the same user preference. Additionally, recent studies (Subramanian, 2023; Hemkumar & Prashanth, 2024) have explored the trade-off between the analysis accuracy and perturbations under DP. Such studies do not necessarily only focus on the privacy benefits of DP, but also aim to maintain high accuracy under noises or perturbations. In summary, inspired by (Wang et al., 2021; Lecuyer et al., 2019; Subramanian, 2023; Hemkumar & Prashanth, 2024), we propose to use DP to augment user profiles for recommendation systems. Our goal is to preserve user preferences during the augmentation process: a perturbed user profile should yield the same recommendations as its original counterpart. This consistency indicates that user preferences are well-preserved in the augmented data, ensuring high-quality positive training samples to compute the NCL losses.

APPENDIX B: EFFICIENT DESIGN

Guarantee with Empirical Expectation. In Theorem 1, the expectation $\mathbb{E}(\mathcal{M}(x)_j)$ is usually estimated via Monte Carlo sampling. That is, one must repeat the inference of the exponential mechanism to draw candidate profiles with different augmentations and compute $\hat{\mathbb{E}}(\mathcal{M}(x)_j) = \frac{1}{m} \sum_{m=1}^m f(x'_m)$. In this appendix, we adapt the proof from Lecuyer et al. (2019) to show that the user preference could still be preserved with approximated empirical expectation. We first obtain an upper bound and a lower bound for $\mathbb{E}(\mathcal{M}(x)_j)$. Specifically, with Hoeffding’s inequality with probability η , we have

$$\begin{aligned}\hat{\mathbb{E}}^{lb}(\mathcal{M}(x)) &= \hat{\mathbb{E}}(\mathcal{M}(x)) - \sqrt{\frac{1}{2m} \ln\left(\frac{2|\mathcal{I}|}{1-\eta}\right)} \\ \hat{\mathbb{E}}^{ub}(\mathcal{M}(x)) &= \hat{\mathbb{E}}(\mathcal{M}(x)) + \sqrt{\frac{1}{2m} \ln\left(\frac{2|\mathcal{I}|}{1-\eta}\right)}\end{aligned}\tag{16}$$

Theorem 2 (Guarantee with Empirical Expectation) Suppose a recommendation mechanism \mathcal{M} satisfies ϵ -DP. For any user sequence x , if \mathcal{M} successfully recommends the ground-truth item y , which is the j -th item within the item scope \mathcal{I} : $y := x_j$, and the predictive score for the ground-truth item is great than the second-largest runner-up score of another item with a small multiplicative factor $e^{2\epsilon}$:

$$\hat{\mathbb{E}}^{lb}(\mathcal{M}(x)_j) > e^{2\epsilon} \max_{k:k \neq j} \hat{\mathbb{E}}^{ub}(\mathcal{M}(x)_k),\tag{17}$$

then for its augmented view x' , with probability higher than η , we have

$$\hat{\mathbb{E}}(\mathcal{M}(x')_j) > \max_{k:k \neq j} \hat{\mathbb{E}}(\mathcal{M}(x')_k).\tag{18}$$

Proof 2

$$\begin{aligned}\hat{\mathbb{E}}(\mathcal{M}(x')_j) &\geq \frac{\hat{\mathbb{E}}(\mathcal{M}(x)_j)}{e^\epsilon} \\ &\geq \frac{\hat{\mathbb{E}}^{lb}(\mathcal{M}(x)_j)}{e^\epsilon}.\end{aligned}\tag{19}$$

Similarly, we have

$$\hat{\mathbb{E}}(\mathcal{M}(x')_k) \leq e^\epsilon \max_{k:k \neq j} \hat{\mathbb{E}}^{ub}(\mathcal{M}(x)_k)\tag{20}$$

Finally, we have

$$\begin{aligned}\hat{\mathbb{E}}(\mathcal{M}(x')_j) &\geq \frac{\hat{\mathbb{E}}(\mathcal{M}(x)_j)}{e^\epsilon} \\ &\geq \frac{\hat{\mathbb{E}}^{lb}(\mathcal{M}(x)_j)}{e^\epsilon} \\ &\geq \frac{e^{2\epsilon} \max_{k:k \neq j} \hat{\mathbb{E}}^{ub}(\mathcal{M}(x)_k)}{e^\epsilon} \\ &= e^\epsilon \max_{k:k \neq j} \hat{\mathbb{E}}^{ub}(\mathcal{M}(x)_k) \\ &\geq \max_{k:k \neq j} \hat{\mathbb{E}}(\mathcal{M}(x')_k).\end{aligned}\tag{21}$$

Exponential Mechanism at Item Level. Recall a critical caveat of designing the exponential mechanism at the user-level is that the search space grows exponentially w.r.t. the length of user histories if the DP augmentation is defined at user-level: $\mathcal{O}(k^l)$. This is because each item $x_i \in x = [x_1, x_2, \dots, x_l]$ has k replacements. As such, there are (k^l) possible candidate user profiles for exponential mechanism to sample from, i.e., $|X'| = k^l$.

Given the sparsity of recommendation data, designing DP augmentations for user sequences introduces prohibitive computation complexity and noises. Therefore, we propose to define the DP augmentation at item level to generate DP augmented user profiles. In this case, the search space grows linearly w.r.t. the length of user histories. To see this, we note that when applying the exponential mechanism to each item $x_i \in x = [x_1, x_2, \dots, x_l]$, the exponential mechanism just need to sample from k candidates, and repeat for all items within the history l times, leading to $\mathcal{O}(k \cdot l)$. Specifically, we compute a DP item for each to-be-replaced item from the original user history using the exponential mechanism. Then, the augmented user profile is generated by replacing the selected items within original user history with the calculated DP items.

To achieve linear complexity, we propose to define the exponential mechanism at item level to generate augmented user profiles, we compute a DP item for each to-be-replaced item from the original user history using the exponential mechanism. Then, the augmented user profile is generated by replacing the selected items within original user history with the calculated DP items. For an item x_i , its scoring function w.r.t. a synonym item x_j is defined as:

$$u(x_i, x_j) = e^{\text{CosSim}(f(x_i), f(x_j))}. \quad (22)$$

Correspondingly, the embedding of the DP item for x_i is computed with

$$\hat{z}_{x_i} = \sum_{x_j \in \mathcal{N}_k(x_i)} \frac{e^{\frac{\epsilon u(x_i, x_j)}{2\Delta_u}}}{\sum_{x_k \in \mathcal{N}_k(x_i)} e^{\frac{\epsilon u(x_i, x_k)}{2\Delta_u}}} \cdot f(x_j). \quad (23)$$

We then replace x_i with the DP item to generated the augmented user profile, which is visualized in Figure 1.

Note this efficient design is feasible and remain DP, because of the two key properties of DP. First, DP has the post-processing property: any computation applied to the output of a DP algorithm remains DP. As such, if the augmentation operation is DP, then recommendation process based on the augmented user profile is also DP. Second, the expected output stability property of DP reduces the computational costs of DP augmentation. This property allows us to generate an expected augmented profile for each user to preserve user preferences in the augmented profiles, instead of generating massive augmented samples for Monte-Carlo sampling.

APPENDIX C: DATASETS STATISTICS

In this section, we provide further details about our datasets. We adopt 6 public benchmark recommendation datasets to evaluate NCL-SR: Beauty, Games, Sports, Toys, Office and Auto He & McAuley (2016); McAuley et al. (2015). These datasets cover different application domains, and are characterized with different sparsity and sequence lengths. Following Yue et al. (2022), we use the 5-core processing to filter out infrequent items and users. The user-item interactions are sorted chronologically to construct training sequences.

To simulate the data sparsity and cold-start users, the split ratio of the training, validation, and test sets is 2:2:6 in our paper. Such a split ratio is selected because we aim to explore the model’s performance under a limited quantity of training data and the model’s generalization on cold-start users, following the recent studies Wu et al. (2024); Qian et al. (2020); Wang et al. (2022a); Lin et al. (2025). For instance, in Lin et al. (2025), there was only 10% training data used to train the model for the recommendation task. However, in our experiments, it is observed that many baselines could not converge under extreme data sparsity. Therefore, we increase the number of training samples for the baselines to converge (20%). This enables a fairer comparison between our method against the baselines. The details datasets statistics are summarized in Table 4.

Datasets	users	Items	Interaction	Length	Density
Beauty	22332	12086	198K	8.88	7e-4
Games	15264	7676	147K	9.69	1e-3
Sports	35265	18173	293K	8.32	5e-4
Toys	19124	11758	165K	8.64	7e-4
Office	4895	2414	53K	10.86	4e-3
Auto	1281	844	8K	6.70	8e-3

Table 4: Overall dataset statistics.

APPENDIX D: ADDITIONAL EXPERIMENTAL RESULTS

In this section, we further investigate the relationship between CL, NCL and data augmentation.

In particular, we first replace the contrastive loss with NCL losses for CL-based baselines. The results are reported in Table 5. In Table 5, each column represent a training scheme being a CL-based baseline or its NCL version. The last column includes the performance of NCL-SR as reference. The comparison is made two-column-wise, with better performance highlighted in bold. In addition, we also make a comparison per row, with the second-best performance highlighted with underlines. From Table 5, we observe that NCL indeed has the potential to improve the recommendation performance while eliminating the need of negative samples for CL-baselines. For instance, the performance of CLS4Rec is generally improved with non-contrastive training over all 6 datasets. For remaining CL-based methods, performance improvement could be observed over half of the datasets. There exists inconsistent performance improvement for certain baselines across different datasets. Such inconsistency is expected, because the data augmentation operations of the CL-baselines are usually ad-hoc and may fail to preserve user preferences. Therefore, the preference inconsistency of the augmented data may degrade the effectiveness of NCL training. Finally, it is observed the performance of NCL-SR is still better than NCL with ad-hoc augmentations. This suggests that our proposed preference-preserving augmentation can indeed better exploit the capacity of NCL training.

Dataset	Metric	CLS4Rec	CLS4Rec + NCL	CoSeRec	CoSeRec + NCL	DUORec	DUORec + NCL	EC4Rec	EC4Rec + NCL	Ours
Beauty	R@10 ↑	0.0690	0.0693	0.0698	0.0683	0.0706	0.0717	0.0628	0.0665	0.0791
	N@10 ↑	0.0367	0.0373	0.0375	0.0373	0.0383	0.0394	0.0344	0.0358	0.0440
	R@20 ↑	0.1008	0.1007	0.1007	0.1023	0.1052	0.1046	0.0971	0.0996	0.1135
	N@20 ↑	0.0447	0.0451	0.0453	0.0458	0.0470	0.0477	0.0430	0.0441	0.0526
Games	R@10 ↑	0.0992	0.1028	0.1057	0.1031	0.1041	0.1002	0.0998	0.1029	0.1140
	N@10 ↑	0.0521	0.0534	0.0553	0.0539	0.0543	0.0520	0.0524	0.0552	0.0611
	R@20 ↑	0.1458	0.1550	0.1560	0.1548	0.1545	0.1519	0.1516	0.1520	0.1683
	N@20 ↑	0.0638	0.0665	0.0680	0.0669	0.0669	0.0650	0.0654	0.0675	0.0748
Sports	R@10 ↑	0.0325	0.0353	0.0327	0.0352	0.0332	0.0364	0.0337	0.0258	0.0441
	N@10 ↑	0.0173	0.0189	0.0169	0.0183	0.0174	0.0189	0.0180	0.0133	0.0237
	R@20 ↑	0.0529	0.0552	0.0501	0.0543	0.0504	0.0543	0.0511	0.0421	0.0660
	N@20 ↑	0.0223	0.0239	0.0213	0.0231	0.0217	0.0234	0.0223	0.0174	0.0292
Toys	R@10 ↑	0.0898	0.0911	0.0868	0.0888	0.0882	0.0876	0.0871	0.0874	0.0941
	N@10 ↑	0.0511	0.0517	0.0490	0.0502	0.0501	0.0481	0.0497	0.0494	0.0537
	R@20 ↑	0.1221	0.1264	0.1199	0.1235	0.1232	0.1206	0.1201	0.1194	0.1286
	N@20 ↑	0.0592	0.0606	0.0573	0.0589	0.0589	0.0564	0.0580	0.0575	0.0623
Office	R@10 ↑	0.0974	0.0970	0.0967	0.1005	0.0974	0.1053	0.0913	0.1012	<u>0.1047</u>
	N@10 ↑	0.0498	0.0512	0.0487	0.0509	0.0518	0.0530	0.0465	0.0503	0.0534
	R@20 ↑	0.1471	0.1484	0.1561	0.1503	0.1497	0.1554	0.1459	0.1452	0.1625
	N@20 ↑	0.0623	0.0642	0.0635	0.0634	0.0647	0.0656	0.0603	0.0614	0.0681
Auto	R@10 ↑	0.1159	0.1256	0.1171	0.1220	0.1098	0.1329	0.1073	0.1305	0.1354
	N@10 ↑	0.0575	0.0630	0.0576	0.0645	0.0572	0.0705	0.0535	0.0629	0.0714
	R@20 ↑	0.1829	0.1878	0.1902	0.1841	0.1927	0.2049	0.1793	0.2012	0.2085
	N@20 ↑	0.0742	0.0786	0.0761	0.0802	0.0781	0.0886	0.0713	0.0808	0.0899

Table 5: Replacing the contrastive loss of CL-based baselines with NCL losses.

Next, we replace the ad-hoc augmentations with our proposed preference-preserving augmentation. In this context, CLS4Rec, CoSeRec and EC4Rec converge to the same CL framework. In comparison, the modified DUORec is also a new baseline, because DUORec has the additional dropout mask. The results are reported in Table 6. In Table 6, each column represent a training scheme being a CL-based baseline or its DP version. The last column includes the performance of NCL-SR as reference. The first comparison is made with first four columns and the second comparison is made with the next two columns, with better performance highlighted in bold. In addition, we also make a comparison per row, with the second-best performance highlighted with underlines. From Table 6, we observe that despite our preference-preserving augmentation, CL-based methods still suffer from generalization issue because of the reliance on negative samples. Due to the limited GPU memory, the training batch size is 4. Under such a small batch size, it is challenging for the CL-based methods to promote uniformity in the learned user/item representations. As a consequence, the trained recommenders may still suffer from generalization issue. Such an observation shows that the per-

formance of CL-based methods is sensitive to negative samples. This is a systematic loophole of CL, which necessitates the development of our NCL framework.

		CLS4Rec	CoSeRec	EC4Rec	+ DP	DUORec	+ DP	Ours
Beauty	R@10 ↑	0.0690	0.0698	0.0628	<u>0.0727</u>	0.0706	0.0674	0.0791
	N@10 ↑	0.0367	0.0375	0.0344	<u>0.0394</u>	0.0383	0.0363	0.0440
	R@20 ↑	0.1008	0.1007	0.0971	<u>0.1077</u>	0.1052	0.0984	0.1135
	N@20 ↑	0.0447	0.0453	0.0430	<u>0.0481</u>	0.0470	0.0440	0.0526
Games	R@10 ↑	0.0992	0.1057	0.0998	0.0989	0.1041	0.1014	0.1140
	N@10 ↑	0.0521	<u>0.0553</u>	0.0524	0.0514	0.0543	0.0529	0.0611
	R@20 ↑	0.1458	0.1560	0.1516	0.1513	0.1545	0.1529	0.1683
	N@20 ↑	0.0638	<u>0.0680</u>	0.0654	0.0645	0.0669	0.0659	0.0748
Sports	R@10 ↑	0.0325	0.0327	0.0337	<u>0.0369</u>	0.0332	0.0358	0.0441
	N@10 ↑	0.0173	0.0169	0.0180	<u>0.0198</u>	0.0174	0.0191	0.0237
	R@20 ↑	0.0529	0.0501	0.0511	<u>0.0553</u>	0.0504	0.0541	0.0660
	N@20 ↑	0.0223	0.0213	0.0223	<u>0.0245</u>	0.0217	0.0237	0.0292
Toys	R@10 ↑	<u>0.0898</u>	0.0868	0.0871	0.0834	0.0882	0.0796	0.0941
	N@10 ↑	<u>0.0511</u>	0.0490	0.0497	0.0456	0.0501	0.0440	0.0537
	R@20 ↑	<u>0.1221</u>	0.1199	0.1201	0.1160	<u>0.1232</u>	0.1141	0.1286
	N@20 ↑	<u>0.0592</u>	0.0573	0.0580	0.0538	0.0589	0.0527	0.0623
Office	R@10 ↑	0.0974	0.0967	0.0913	0.0958	0.0974	<u>0.0980</u>	0.1047
	N@10 ↑	0.0498	0.0487	0.0465	0.0495	0.0518	<u>0.0527</u>	0.0534
	R@20 ↑	0.1471	0.1561	0.1459	0.1398	0.1497	0.1465	0.1625
	N@20 ↑	0.0623	0.0635	0.0603	0.0606	<u>0.0647</u>	<u>0.0647</u>	0.0681
Auto	R@10 ↑	0.1159	0.1171	0.1073	<u>0.1317</u>	0.1098	0.1268	0.1354
	N@10 ↑	0.0575	0.0576	0.0535	0.0644	0.0572	<u>0.0678</u>	0.0714
	R@20 ↑	0.1829	0.1902	0.1793	0.2000	0.1927	<u>0.2073</u>	0.2085
	N@20 ↑	0.0742	0.0761	0.0713	0.0814	0.0781	<u>0.0879</u>	0.0899

Table 6: Applying DP augmentation for CL-based methods. The best results are highlighted in bold and the second best results are highlighted with underline.

APPENDIX E: ADDITIONAL SENSITIVITY ANALYSIS

Sensitivity Analysis w.r.t. Alignment. In the alignment loss (Equation 10), there exists an additional hyperparameter γ that affects the computation of \mathcal{L}_{align} . Firstly, we acknowledge that different values of γ could indeed affect the model performance. This is revealed in our sensitivity analysis w.r.t. λ_2 as in Figure 2b. To see this, note that λ_2 is equivalent to a rescaled γ in our implementation. As shown in [1], $\mathcal{L}_{uniform} + \mathcal{L}_{align}$ could be expanded as follows:

$$\begin{aligned}
& \mathcal{L}_{uniform}(Z, Z') + \mathcal{L}_{align}(Z, Z') \\
&= \text{MCE}\left(\frac{1}{d}I_d, C(Z, Z')\right) - \text{tr}(C(Z, Z')) + \gamma \cdot \text{MCE}(C(Z, Z), C(Z', Z')) \\
&= -\text{tr}\left(\left(\frac{1}{d}I_d\right)\log(C(Z, Z'))\right) - \gamma \cdot \text{tr}(C(Z, Z)\log(C(Z', Z'))) + \gamma \cdot \text{tr}(C(Z', Z')) + \text{const.}
\end{aligned} \tag{24}$$

In the last expansion in the third row, it is observed that both the second term and the third term are derived from \mathcal{L}_{align} . More importantly, both these two terms are re-scaled by γ . Since the second term and third term correspond to \mathcal{L}_{align} , we can instead use another scaling factor that directly adjusts the trade-off between $\mathcal{L}_{uniform}$ and \mathcal{L}_{align} . Specifically, we used λ_1 to re-scale $\mathcal{L}_{uniform}$ and λ_2 to re-scale \mathcal{L}_{align} with γ being absorbed into λ_2 . Finally, such an implementation (i.e., absorbing γ into λ_2) reduces the number of hyperparameters required for tuning.

Additional Sensitivity Analysis w.r.t. Number of Perturbations. We conduct a sensitivity analysis w.r.t. the number of perturbations in the user profile generation. Recall, the preference-preserving user profiles are generated by injecting item-level perturbations into the original user profile. In this set of experiments, we increase the number of perturbations from 1 to 8. We stopped at 8, because the averaged length of user profiles in most datasets is around 8. We report the results of sensitivity analysis in Table 7. According to Table 7, it is observed that in general, the model performance is stable when varying the number of perturbations from 1 to 4. Moreover, the performance of the model starts degrading after the number of perturbations is larger than 5. This is expected, because, with too many perturbations, the initial model may generate some wrong user preferences at the beginning. During training, such wrong user preferences are preserved, so the model is then trained to learn wrong user preferences, which degrades the performance.

Dataset	Num. of Perturbations	1	2	3	4	5	6	7	8
Beauty	R@10 ↑	0.0772	0.0779	0.0791	0.0742	0.0783	0.0749	0.0726	0.0747
	N@10 ↑	0.0415	0.0427	0.0440	0.0396	0.0430	0.0408	0.0400	0.0404
	R@20 ↑	0.1130	0.1139	0.1135	0.1098	0.1165	0.1141	0.1089	0.1113
	N@20 ↑	0.0505	0.0517	0.0526	0.0486	0.0526	0.0507	0.0492	0.0496
Games	R@10 ↑	0.1089	0.1123	0.1140	0.1101	0.1076	0.1107	0.1104	0.1085
	N@10 ↑	0.0574	0.0600	0.0611	0.0572	0.0561	0.0582	0.0573	0.0583
	R@20 ↑	0.1656	0.1664	0.1683	0.1650	0.1626	0.1647	0.1629	0.1607
	N@20 ↑	0.0716	0.0736	0.0748	0.0710	0.0700	0.0717	0.0705	0.0714
Sports	R@10 ↑	0.0397	0.0387	0.0441	0.0406	0.0382	0.0351	0.0358	0.0394
	N@10 ↑	0.0210	0.0206	0.0237	0.0209	0.0201	0.0191	0.0192	0.0212
	R@20 ↑	0.0593	0.0592	0.0660	0.0617	0.0596	0.0565	0.0549	0.0611
	N@20 ↑	0.0259	0.0258	0.0292	0.0262	0.0255	0.0244	0.0240	0.0266
Toys	R@10 ↑	0.0930	0.0958	0.0941	0.0942	0.0926	0.0908	0.0922	0.0928
	N@10 ↑	0.0511	0.0540	0.0537	0.0540	0.0525	0.0515	0.0521	0.0522
	R@20 ↑	0.1248	0.1309	0.1286	0.1314	0.1281	0.1238	0.1261	0.1302
	N@20 ↑	0.0592	0.0628	0.0623	0.0634	0.0615	0.0598	0.0607	0.0615
Office	R@10 ↑	0.0983	0.0989	0.1047	0.1028	0.0983	0.0875	0.0989	0.0980
	N@10 ↑	0.0521	0.0524	0.0534	0.0529	0.0506	0.0431	0.0498	0.0509
	R@20 ↑	0.1561	0.1535	0.1625	0.1570	0.1554	0.1459	0.1519	0.1551
	N@20 ↑	0.0662	0.0662	0.0681	0.0665	0.0649	0.0579	0.0631	0.0653
Auto	R@10 ↑	0.1390	0.1280	0.1354	0.1268	0.1220	0.1268	0.1256	0.1220
	N@10 ↑	0.0690	0.0660	0.0714	0.0667	0.0630	0.0677	0.0672	0.0626
	R@20 ↑	0.2207	0.2098	0.2085	0.2037	0.2049	0.1976	0.2024	0.1902
	N@20 ↑	0.0894	0.0867	0.0899	0.0860	0.0839	0.0856	0.0863	0.0800

Table 7: Changing number of perturbations for preference-preserving profile generation.

APPENDIX F: IMPLEMENTING ID-BASED SR MODELS WITH E5

We further implement SR models with E5, where E5 is used as the embedding model to embed the item texts. We compare our method against the E5-based SR models in the table below. It is observed that our method can outperform the E5-based SR models. We also note that it is non-trivial to apply E5 to these models, because such ID-based models originally take item IDs as inputs, whereas E5 is a text-based model.

Dataset	Metric	E5-Based				
		BERT4Rec	SASRec	NARM	LRURec	NCL-SR (Ours)
Beauty	R@10 ↑	0.0420	<u>0.0504</u>	0.0348	0.0446	0.0791
	N@10 ↑	0.0226	<u>0.0249</u>	0.0189	0.0210	0.0440
	R@20 ↑	0.0672	<u>0.0784</u>	0.0486	0.0614	0.1135
	N@20 ↑	0.0290	<u>0.0319</u>	0.0224	0.0292	0.0526
Games	R@10 ↑	0.0739	<u>0.1061</u>	0.0422	0.0641	0.1140
	N@10 ↑	0.0385	<u>0.0556</u>	0.0236	0.0351	0.0611
	R@20 ↑	0.1150	<u>0.1568</u>	0.0563	0.0946	0.1683
	N@20 ↑	0.0488	<u>0.0682</u>	0.0272	0.0429	0.0748
Sports	R@10 ↑	<u>0.0287</u>	0.0253	0.0105	0.0164	0.0441
	N@10 ↑	<u>0.0159</u>	0.0126	0.0064	0.0095	0.0237
	R@20 ↑	<u>0.0431</u>	0.0372	0.0144	0.0228	0.0660
	N@20 ↑	<u>0.0195</u>	0.0156	0.0074	0.0111	0.0292
Toys	R@10 ↑	0.0379	<u>0.0533</u>	0.0127	0.0198	0.0941
	N@10 ↑	0.0197	<u>0.0268</u>	0.0088	0.0133	0.0537
	R@20 ↑	0.0576	<u>0.0739</u>	0.0158	0.0256	0.1286
	N@20 ↑	0.0247	<u>0.0320</u>	0.0096	0.0147	0.0623
Office	R@10 ↑	0.0342	0.0307	0.0425	<u>0.0466</u>	0.1047
	N@10 ↑	0.0170	0.0156	<u>0.0210</u>	<u>0.0210</u>	0.0534
	R@20 ↑	0.0549	0.0434	<u>0.0836</u>	0.0782	0.1625
	N@20 ↑	0.0222	0.0187	<u>0.0313</u>	0.0289	0.0681
Auto	R@10 ↑	0.0378	0.0439	0.0402	<u>0.0512</u>	0.1354
	N@10 ↑	0.0206	<u>0.0279</u>	0.0207	0.0275	0.0714
	R@20 ↑	0.0646	<u>0.0537</u>	0.0683	<u>0.0817</u>	0.2085
	N@20 ↑	0.0282	0.0303	0.0227	<u>0.0352</u>	0.0899

Table 8: Implementing SR models with E5.

APPENDIX G: COMPUTATIONAL COSTS AND MEMORY CONSUMPTION BETWEEN CL AND NCL

From a theoretical point of view, the time complexity of CL is $\mathcal{O}(n^2)$, if the training batch size is n and the loss is computed with in-batch negative samples. In comparison, the time complexity of NCL is $\mathcal{O}(n)$, because there are no negative samples at all. Therefore, NCL is more computationally efficient than CL (Zhuo et al., 2023; Zbontar et al., 2021; Grill et al., 2020; Cho et al., 2022).

Furthermore, we highlight that in our method, the computation of the logarithm in Equation 8 is approximated with Taylor expansion as in (Zhang et al., 2023). In general, the time complexity of this operation is the time complexity is $\mathcal{O}(md^2)$ (under our implementation), with m being the order of the expansion and d being the dimensionality of the matrix V . In our implementation, we set $m = 4$ for computational efficiency and $d = 768$ as given by the dimensionality of E5 outputs:

$$\log V = \sum_{i=1}^m (-1)^{(m+1)} \frac{(V - I)^m}{m}.$$

Finally, we conducted additional experiments to evaluate the computational cost and memory consumption of computing CL and NCL losses. In this set of experiments, we ensure that all methods use exactly the same training configuration (i.e., batch size, model, CUDA version). The hardware used to evaluate memory consumption is NVIDIA A40. We report the results in the table

below. In the Table below, we report the relative memory consumption between NCL and CL:
 $\eta_{memory} = \frac{\text{memory consumption of NCL}}{\text{memory consumption of CL}}:$

Batch Size	1	2	3	4	5
η_{memory}	0.72	0.57	0.51	0.51	CL out of memory

Table 9: Memory consumption for computing CL and NCL losses with different batch sizes.