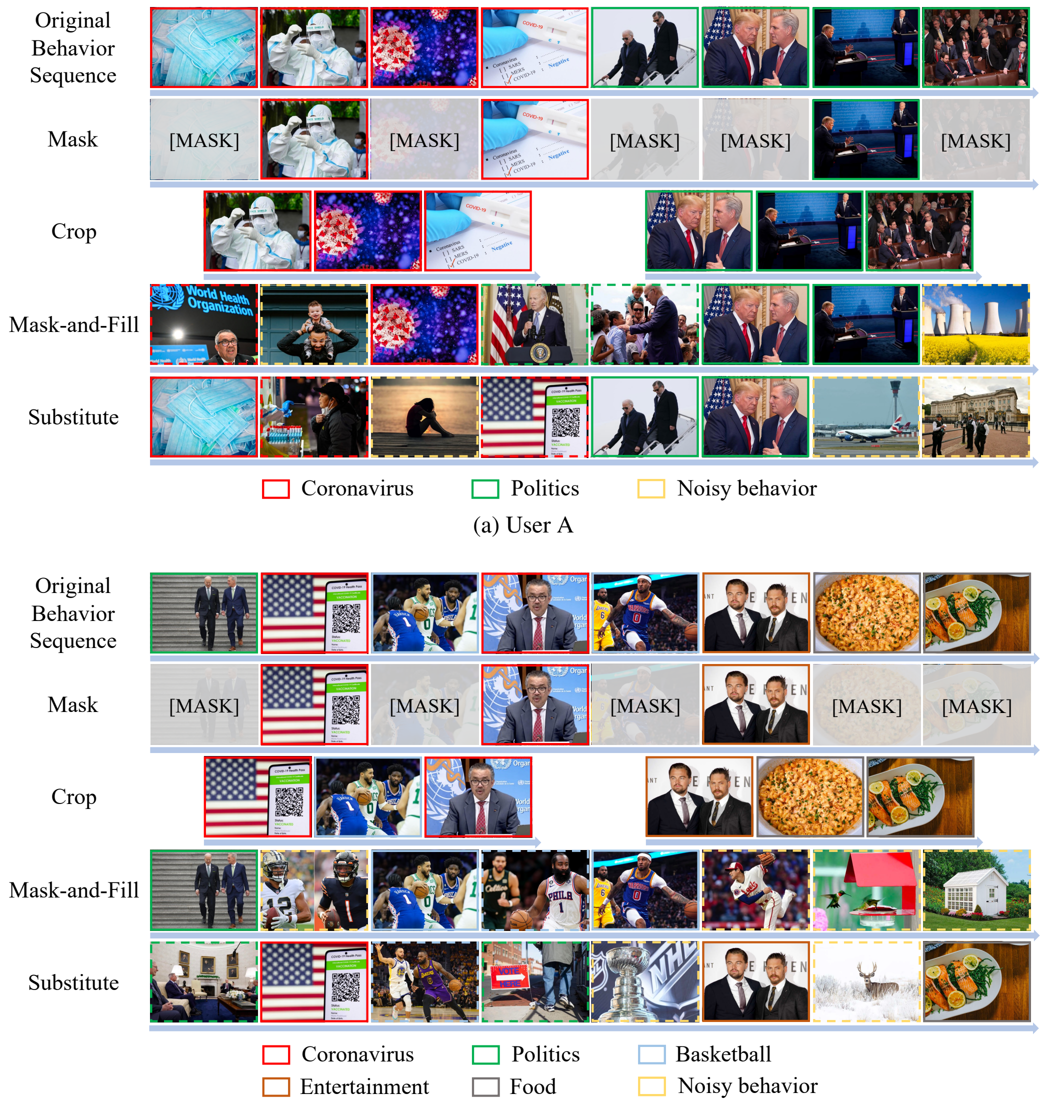
A More Examples of the Semantic Inconsistency Problem



(a) User A

(b) User B

Figure 1: Illustrations of the impact of diverse data augmentation methods on different user behavior sequences. Each picture represents a news article clicked by the user. Pictures with the same color border reflect similar interests. Dash borders indicate behaviors replaced by the augmentation method.

We randomly sample two more anonymous users from the online news platform to illustrate the impact of various data augmentation methods on different user behavior sequences and the consequent semantic inconsistency problem. The users' recent eight behaviors and the results of several data augmentation methods are shown in Fig. 1. The data augmentation proportion is set as 0.6. From the original behavior sequence of user A, we may infer that the user is quite concerned about coronavirus and politics. We also find that the behavior sequence augmented by masking well preserves the user's interests. However, the two sequences augmented by cropping contain completely different interests of the user over time. Meanwhile, some behaviors replaced by mask-and-fill and substitute are noisy, which the user may not be interested in. Similarly, the original behavior sequence of user B shows the user's potential interests in coronavirus, politics, basketball, entertainment, and food. As a result, augmentation methods such as mask and crop are likely to lose certain interests of the user, while augmentation methods such as mask-and-fill and substitute tend to bring in noisy behaviors. In fact, it is hard to determine what are the real interests of user B from such a diverse behavior sequence since some behaviors may be the result of misclicking or click-baiting, thus making it even harder to provide a semantically consistent augmentation. From these examples, we can find that existing data augmentation methods may fail to preserve the characteristics or interests in the behavior sequence and cannot guarantee semantic consistency between the augmented views. Thus, directly forcing the user model to maximize the agreement between the augmented sequences may result in a negative transfer for downstream tasks.

B The Algorithm of AdaptSSR

The pseudo-codes of the pre-training procedure with our AdaptSSR are shown in Algorithm 1.

Algorithm 1 Pre-training Procedure with AdaptSSR

Input: A corpus of user behavior sequences $\mathcal{S}$ and a set of data augmentation operators $\mathcal{A}$.
Output: The pre-trained user model $\mathcal{M}$.

Randomly initialize the parameter of the user model $\mathcal{M}$.
while not converged **do**
    Randomly sample a batch of user behavior sequences $\{S_i\}_{i=1}^{B}$ from $\mathcal{S}$.
    for each S_i **do**
        Randomly select two augmentation operators f and g from $\mathcal{A}$.
        $\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i^+ \leftarrow \mathcal{M}(f(S_i)), \mathcal{M}(f(S_i))$. ▷ With independently sampled dropout masks.
        $\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_i^+ \leftarrow \mathcal{M}(g(S_i)), \mathcal{M}(g(S_i))$. ▷ With independently sampled dropout masks.
        $\mathbf{U}_i^- \leftarrow \{\hat{\boldsymbol{u}}_j, \hat{\boldsymbol{u}}_j^+, \tilde{\boldsymbol{u}}_j, \tilde{\boldsymbol{u}}_j^+\}_{j=1, j\neq i}^{B}$.
        $\lambda_i \leftarrow 1 - \frac{1}{4}\sum_{\hat{\boldsymbol{s}}\in\{\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i^+\}}\sum_{\tilde{\boldsymbol{s}}\in\{\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_i^+\}} \mathrm{sim}(\hat{\boldsymbol{s}}, \tilde{\boldsymbol{s}})$.
        $$\hat{\mathcal{L}}_i \leftarrow -\log\sigma\Big[\lambda_i\left(\mathrm{sim}\left(\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i^+\right) - \max_{\boldsymbol{v}\in\{\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_i^+\}}\mathrm{sim}\left(\hat{\boldsymbol{u}}_i, \boldsymbol{v}\right)\right) + (1-\lambda_i)\left(\min_{\boldsymbol{v}\in\{\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_i^+\}}\mathrm{sim}\left(\hat{\boldsymbol{u}}_i, \boldsymbol{v}\right) - \max_{\boldsymbol{w}\in\mathbf{U}_i^-}\mathrm{sim}\left(\hat{\boldsymbol{u}}_i, \boldsymbol{w}\right)\right)\Big].$$
        $$\tilde{\mathcal{L}}_i \leftarrow -\log\sigma\Big[\lambda_i\left(\mathrm{sim}\left(\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_i^+\right) - \max_{\boldsymbol{v}\in\{\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i^+\}}\mathrm{sim}\left(\tilde{\boldsymbol{u}}_i, \boldsymbol{v}\right)\right) + (1-\lambda_i)\left(\min_{\boldsymbol{v}\in\{\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i^+\}}\mathrm{sim}\left(\tilde{\boldsymbol{u}}_i, \boldsymbol{v}\right) - \max_{\boldsymbol{w}\in\mathbf{U}_i^-}\mathrm{sim}\left(\tilde{\boldsymbol{u}}_i, \boldsymbol{w}\right)\right)\Big].$$
    end for
    $\mathcal{L} \leftarrow \sum_{i=1}^{B}\left(\hat{\mathcal{L}}_i + \tilde{\mathcal{L}}_i\right)/2B$.
    Update the parameters of the user model $\mathcal{M}$ with $\mathcal{L}$ by backpropagation.
end while

C Implementation Details

In our experiments, the embedding dimension d is set as 64. In the Transformer Encoder, the number of attention heads and layers are both set as 2. We only utilize the standard dropout mask, which is applied to both the attention probabilities and the output of each sub-layer, and the dropout probability is set as 0.1. The maximum sequence length is set to 100 and 256 for the TTL dataset and the App dataset, respectively. For each downstream task, a two-layer MLP is added to the pre-trained user model, and the dimension of the intermediate layer is also set as 64.

For existing pre-training methods, the data augmentation proportion ρ is either searched from $\{0.1, 0.2, \ldots, 0.9\}$ or copied from previous works if provided. In our AdaptSSR, we adopt three random augmentation operators by default: mask, crop, and reorder. The augmentation proportion is set to 0.6, 0.4, and 0.6 respectively while our experimental results have shown that our method is robust to various data augmentation methods with different strengths.

We use the Adam optimizer for model training. The batch size and learning rate are set as 128 and 2e-4 for both pre-training and fine-tuning. The test results for all the models are reported at their best validation epoch. We repeat each experiment five times with different random seeds and report the average results. We implement all experiments with Python 3.8.13 and Pytorch 1.12.1 on an NVIDIA Tesla V100 GPU. Our code is available at `https://anonymous.4open.science/r/AdaptSSR/`.

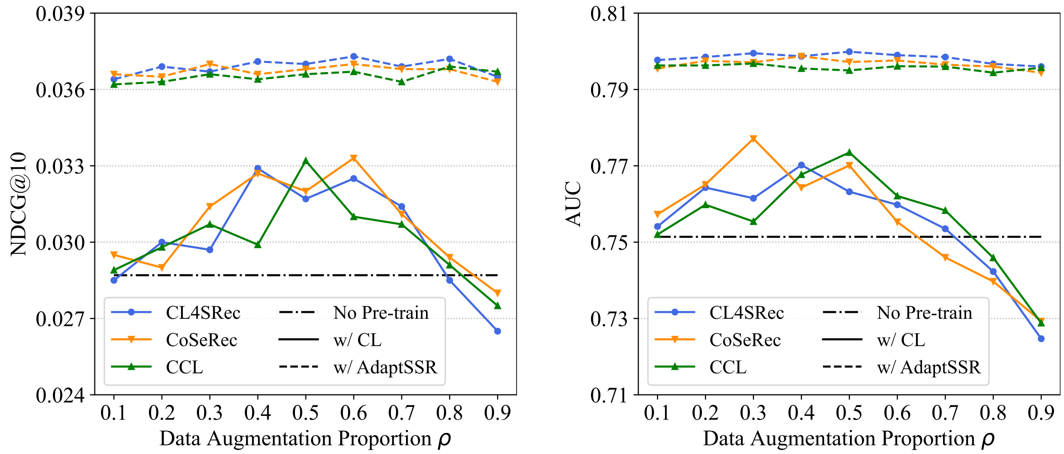


Figure 2: The effectiveness of AdaptSSR when combined with existing pre-training methods on the downstream thumb-up recommendation task (left) and CVR prediction task (right).

D More Experimental Results

Fig. 2 shows the performance of AdaptSSR when combined with several existing pre-training methods: CL4SRec, CoSeRec, and CCL, on the downstream thumb-up recommendation task ($\mathcal{T}_4$) and CVR prediction task ($\mathcal{T}_6$) with the data augmentation proportion ρ varying from 0.1 to 0.9. Similar to the results on the age prediction task ($\mathcal{T}_1$), we find that these contrastive learning-based methods are highly sensitive to the data augmentation proportion. A too-small or too-large value of ρ will lead to limited performance gain or even negative transfer for the downstream task. In contrast, our AdaptSSR consistently boosts the effectiveness of these pre-training methods by a large margin. This is because our self-supervised ranking task avoids directly maximizing the similarity between the augmented views. Moreover, our AdaptSSR also substantially improves the robustness of these pre-training methods to the augmentation proportion, which verifies the effectiveness of our augmentation-adaptive fusion mechanism. The dynamic coefficient λ_i enables the user model to automatically combine the learned pairwise ranking orders based on the similarity between the augmented views for each training sample, thus empowering the pre-training methods to adapt to divse data augmentation methods with varying strengths.

E Limitations

In this work, after pre-training the user model with our AdaptSSR, the entire model is fine-tuned with the downstream labeled data. However, fine-tuning such a large model for each downstream task can be time- and space-consuming. A potential solution for this problem is Parameter-Efficient Fine-Tuning (PEFT). Several methods such as Adapter and LoRA have been demonstrated to be effective for adapting the large language model to various downstream tasks by only updating a small fraction of parameters. However, we empirically find that these methods perform poorly when applied to the pre-trained user model. A possible reason is that the currently widely used Transformer-based user model is much shallower and thinner than the large language model. Most of the model parameters belong to the bottom behavior embedding table, while the upper Transformer blocks only contain very few parameters. Since existing PEFT methods mainly focus on tuning parts of parameters in each Transformer block, only a very small fraction of parameters in total will be updated, which may not be enough to effectively adapt the user model to the downstream task. We will investigate how to transfer the pre-trained user model to various downstream tasks parameter-efficiently while maintaining the performance gain brought by AdaptSSR in our future work.