A APPENDIX

A.1 Implementation Details and Motivations

AFT mainly consists of a generator and a discriminator. We use transformer extractors in both generator and discriminator since they can effectively learn feature interactions between different domains. In the two-step feature translation, inspired by the knowledge representation learning method that learns with triples *(head entity, relation, tail entity)*, we also use similar feature translations via *(item-level preference, domain-level preference, user general preference)* and *(user general preference, domain information, user domain-specific preference)* to model interpretable relations between item, domain and user. We use ConvE since it is efficient and effective. Other KRL models could also be easily adopted in the two-step feature translation. All item and domain embeddings are randomly initialized. Note that the "translation" in Adversarial feature translation indicates that "the process of moving something from one place to another", not limited to mathematical translation with the strict linear transformation.

In Sec. 4.2, we bring in detailed discussions on the training challenges and solutions of AFT. The MMD loss aims to broaden the distance between the expectations of fake clicked and real clicked item embedding distributions, which has been widely used in other works. The key hyper-parameter to keep a fast and stable convergence is the number of fake clicked items provided by the generator. We combine these generated items with real unclicked items as negative samples. We have tested the fake clicked item numbers among $\{1, 3, 5, 10, 15, 20\}$, and select the best 5 in the final version. For model parameters, we introduce most of the essential hyper-parameters in Sec. 5.3, including embedding dimensions, learning parameters and weights. We use a grid search for parameter selection. Note that we cannot choose too large embedding dimensions for our model, considering the online memory and computation costs. The final parameters will be listed in the released version of the source codes. We implement AFT with multi-core GPU, using Tensorflow 1.4.1 with Linux. The amount of memory is 60G.

A.2 Evaluation Details

In offline evaluation, we conduct five runs and report the average results. We also conduct significance tests to verify that the improvements are significant. For the evaluation metric, we mainly concentrate on Recall@N since we deploy AFT on the matching module, which cares whether good items are retrieved in top N items (N is usually set as several hundred in real-world large-scale recommendation systems). We use Recall@50 and Recall@200 for WeChat MDR dataset, and Recall@20 and Recall@50 for Netflix. Similar evaluation with Recall@N on Netflix is also reported in Shenbin et al. [21]. We also consider AUC as a supplement to evaluate the overall ranking ability. We select several strong baselines of all four different method types and report their AUC results in three domains. We highlight that AFT achieves the best performances on all five domains of WeChat measured by Recall@N and AUC. In general, the conclusions of AUC and Recall@N are basically consistent.

In ablation study, *AFT - generator* indicates the AFT version that only uses the discriminator with real unclicked items as negative samples, which is equivalent to the AFT (D) version. *AFT - domain-specific mask* indicates that AFT keeps the generator but removes the domain-specific masked encoder. *AFT - feature translation* indicates that we replace the two-step feature translation with a domain-specific attention-based aggregation. *AFT - dissimilarity loss* indicates that we only use L_D and L_G in training. Other components remain unchanged in all ablation settings.