AMEIR: Automatic Behavior Modeling, Interaction Exploration and MLP Investigation in the Recommender System

Pengyu Zhao\textsuperscript{1,2}, Kecheng Xiao\textsuperscript{1}, Yuanxing Zhang\textsuperscript{3}, Kaigui Bian\textsuperscript{1} and Wei Yan\textsuperscript{1}

\textsuperscript{1}School of EECS, Peking University, Beijing, China
\textsuperscript{2}HULU LLC, Beijing, China
\textsuperscript{3}Alibaba Group, Beijing, China

\{pengyuzhao, kecheng, bkg, w\}@pku.edu.cn, yuanxing.zyx@alibaba-inc.com

Abstract

Recently, deep learning models have been widely spread in the industrial recommender systems and boosted the recommendation quality. Though having achieved remarkable success, the design of task-aware recommender systems usually requires manual feature engineering and architecture engineering from domain experts. To relieve those human efforts, we explore the potential of neural architecture search (NAS) and introduce AMEIR for Automatic behavior Modeling, interaction Exploration and multi-layer perceptron (MLP) Investigation in the Recommender system. The core contributions of AMEIR are the three-stage search space and the tailored three-step searching pipeline. Specifically, AMEIR divides the complete recommendation models into three stages of behavior modeling, interaction exploration, MLP aggregation, and introduces a novel search space containing three tailored subspaces that cover most of the existing methods and thus allow for searching better models. To find the ideal architecture efficiently and effectively, AMEIR realizes the one-shot random search in recommendation progressively on the three stages and assembles the search results as the final outcome. Further analysis reveals that AMEIR’s search space could cover most of the representative recommendation models, which demonstrates the universality of our design. The extensive experiments over various scenarios reveal that AMEIR outperforms competitive baselines of elaborate manual design and leading algorithmic complex NAS methods with lower model complexity and comparable time cost, indicating efficacy, efficiency and robustness of the proposed method.

1 Introduction

Recommender system has become an essential service on online E-commerce business and content platforms to deliver items that best fit users’ interests from the substantial number of candidates. Recently, advancements of deep learning (DL) have innovated the recommending strategies in real-world applications [Covington et al., 2016; Zhou et al., 2018b] to enable accurate and personalized recommendation.

DL-based recommender systems have to process both sequential and non-sequential input features, and thereby follow a characterized architecture compared to the general DL tasks. A canonical DL-based recommendation model usually includes three parts: (1) Behavior modeling [Hidasi and Karatzoglou, 2018; Kang and McAuley, 2018; Tang and Wang, 2018; Zhou et al., 2018b] probes into capturing user’s diverse and dynamic interest from historical behaviors; (2) Interaction exploration [Guo et al., 2017; Lian et al., 2018; Qu et al., 2018; Wang et al., 2017] finds useful interactions among different fields to provide memorization and intuitive conjunction evidence for the recommendation. (3) Multi-layer perceptron (MLP) investigation [Covington et al., 2016; Zhou et al., 2018b] aggregates the inputs with the results from behavior modeling and interaction exploration, and then fuses the features with hidden layers in the MLP.

Industry demands efficiently designing the three parts in the DL-based recommendation model to improve the accuracy and personalization for given recommender tasks in business. Nevertheless, it is hard to find a unified model meeting the requirements in all the scenarios. For the models of behavior modeling, recurrent neural network (RNN) [Hidasi and Karatzoglou, 2018; Hidasi et al., 2016] is hard to preserve long-term behavioral dependencies even though employing gated memory cells. Convolutional neural network (CNN) [Tang and Wang, 2018; Yuan et al., 2019] is capable of learning local feature combinations yet it relies on a wide kernel, deep network or global pooling to distinguish long-term interests. The attention mechanism [Zhou et al., 2018b] directly aggregates the entire behavior sequence, but it cannot capture the evolution of the user’s preference [Zhou et al., 2019]. Self-attention [Vaswani et al., 2017] is better for modeling long-term behaviors [Feng et al., 2019; Sun et al., 2019], but it is hard to be deployed in real-time applications that require fast inference with limited resources. Regarding the methods for interaction exploration and MLP investigation, vanilla MLP [Covington et al., 2016] implicitly generalizes high-order information in the network whereas missing the low-order information. Therefore, linear regression (LR) [Cheng et al., 2016], factorization-
tion machine (FM) [Guo et al., 2017; He and Chua, 2017] and product-based neural network (PNN) [Qu et al., 2016; Qu et al., 2018] are introduced to involve low-order feature interactions from raw input categorical features. On top of that, DCN [Wang et al., 2017] and xDeepFM [Lian et al., 2018] further learn compressed high-order interactions in the networks. However, these methods either require hand-crafted cross features or simply enumerate interactions of bounded degree, and thus introducing noise in the model [Liu et al., 2020]. Moreover, the MLP specified manually or by grid search is usually sub-optimal. Therefore, adopting and coordinating the task-aware architectures for the three parts of recommendation models are the main challenge for building efficient and accurate recommender systems. 

Recently, the neural architecture search (NAS) paradigm [Zoph et al., 2018; Liu et al., 2019; Xie et al., 2019; Real et al., 2019; Tan and Le, 2019; So et al., 2019; Wang et al., 2020b] is proposed to automatically design deep learning models by searching task-specific and data-oriented optimal architecture from the search space, thereby mitigating a lot of human efforts. Some latest attempts [Luo et al., 2019; Liu et al., 2020; Song et al., 2020] incorporate NAS with the recommendation. Nevertheless, they all neglect behavior modeling and only consider interaction exploration and MLP investigation in the restricted search spaces. Different from these methods, we introduce AMEIR, namely Automatic behavior Modeling, interaction Exploration and MLP Investigation in the Recommender system, to automatically find the complete recommendation models for adapting various recommender tasks and mitigating manual efforts. Specifically, AMEIR divides the backbone architecture of recommendation models based on the three stages of behavior modeling, interaction exploration, MLP aggregation, and introduces a novel search space containing three tailored subspaces that are designed to cover most of the representative recommender systems and thus allow for searching better architectures. Facing the industrial demands of agile development and architecture iteration, AMEIR realizes the efficient while competitive one-shot random search [Li and Talwalkar, 2019] in this paper, and proposes a three-step searching pipeline to progressively find the ideal architectures for the corresponding three stages in the recommendation model. The search results will be assembled as the final outcome. AMEIR would not add complicated or redundant operations to the computation graph, and thus the yielded models can be optimized by the common industrial accelerating strategies [NVIDIA, 2020]. The experimental results over various recommendation scenarios show that our automatic paradigms consistently achieves state-of-the-art performance despite the influence of multiple runs, and could outperform both the strongest baselines with elaborate manual design and leading NAS methods with comparable search cost, indicating efficacy, efficiency, and robustness of the proposed method. Further analysis reveals that the overall time cost of AMEIR is similar to the baseline methods without architecture search and the searched models always possess lower model complexity, which demonstrates that AMEIR is efficient in both search and inference phases.

2 AMEIR

In Sec.2, we will introduce the three-stage search space, and the tailored three-step one-shot searching pipeline in AMEIR. We will also discuss the relationship between AMEIR and representative recommender systems and show that AMEIR is able to cover most of these methods.

2.1 Backbone Model in AMEIR’s Search Space

The recommendation models in AMEIR’s search space share the same three-stage backbone, as illustrated in Fig.1. Stage 0: Feature Representation and Embedding. In the common recommender systems, the input features are collected in multi-field categorical form, and generally, there are three basic elements in the data instances: user profile, item profile and context profile. User profile represents the user’s personal information and can be expressed by a tuple of multiple fields, e.g., ("User ID", "User Attributes", etc). Analogously, item profile and context profile can be represented by ("Item ID", "Item Attributes", etc) and ("Time", "Place", etc). Based on the basic elements, AMEIR segregates the features into two groups: segregated into sequential features and non-sequential features: (1) The sequential features describe the user’s historical behaviors, which can be represented by a sequence of behavior elements. Each behavior element contains the corresponding item profile and the optional context profile. (2) The non-sequential features depict the attribute information of the current recommendation, including user profile, instant context profile, and the optional target profile for click-through rate (CTR) prediction task. Conforming to [Covington et al., 2016; Zhou et al., 2018b], AMEIR transforms the data instances into high-dimensional sparse vectors via one-hot or multi-hot encoding. Specifically, if the field is univalent, it will be converted to one-hot vector; if the field is multivalent, it will be encoded as a multi-hot vector. Each data instance can be formally illustrated as:

\[
\begin{aligned}
\mathbf{x} &= [\mathbf{x}^a, \mathbf{x}^b] = [x^a_1, x^a_2, ..., x^a_T, x^b_1, x^b_2, ..., x^b_N]
\end{aligned}
\]  

where \(x^a = [x^a_1, x^a_2, ..., x^a_T]\) denotes the encoded sequential behavior elements of length \(T\) with each behavior element \(x^a_t = [x^a_{i,1}, x^a_{i,2}, ..., x^a_{i,N_a}]\) grouped in \(N_a\) categorical fields of the same dimension, and \(x^b = [x^b_1, x^b_2, ..., x^b_N]\) represents \(N\) encoded vectors of non-sequential feature fields. AMEIR compresses the sparse features into low dimensional dense vectors via embedding technique. Suppose there are \(F\) unique features, AMEIR creates the embedding matrix \(\mathbf{E} \in \mathbb{R}^{F \times K}\), where each embedding vector has dimension \(K\). The embedding process follows the embedding lookup mechanism: If the field is univalent, the field embedding is the feature embedding; If the field is multivalent, the field embedding is the sum of feature embedding. The embedded data instance can then be represented by:

\[
\begin{aligned}
\mathbf{e} &= [\mathbf{e}^a, \mathbf{e}^b] = [e^a_1, e^a_2, ..., e^a_T, e^b_1, e^b_2, ..., e^b_N]
\end{aligned}
\]  

where \(e^a_t = [e^a_{i,1}, e^a_{i,2}, ..., e^a_{i,N_a}] \in \mathbb{R}^{N_a \times K}\) is the concatenation of the embeddings in \(x^a_t = [x^a_{i,1}, x^a_{i,2}, ..., x^a_{i,N_a}]\), and \(e^b_t \in \mathbb{R}^{K}\) is the embedding of \(x^b_t\).
Stage 1: Behavior Modeling on Sequential Features. To extract the sequential representation from user’s behavior, AMEIR introduces a block-wise behavior modeling network to model sequential patterns based on grouped behavioral embedding $H^b_0 = [e^b_1, e^b_2, ..., e^b_T]$. The behavior modeling network is constructed by stacking a fixed number of $L_b$ residual building blocks, where the $l$-th block receives the hidden state $H^b_{l-1}$ from the previous block as input and generates a new hidden state $H^b_l$ of the same dimension. Each block consists of three consecutive operations of normalization, layer, and activation, which are selected from the corresponding operation sets in the search subspace of building blocks. The hidden state of layer $l$ is formulated as:

$$H^b_l = \text{Act}^b_l(\text{Layer}^b_l(\text{Norm}^b_l(H^b_{l-1}))) + H^b_{l-1},$$

where $\text{Act}^b_l$, $\text{Layer}^b_l$ and $\text{Norm}^b_l$ are the selected activation, layer and normalization operations for layer $l$. As the fully connected networks can only handle fixed-length inputs, AMEIR compresses the final hidden states into fixed dimensions via sequence-level sum pooling: $h^b_{pool} = \sum_{t=1}^T h^b_{l,a,i}$, where $h^b_{l,a,i}$ denotes the $i$-th element in $H^b_l$. Moreover, to extract the latest interest of the user, the last hidden state $H^b_{L_b}$ is also included in the result. Besides, when the target item is involved, AMEIR adopts an attention layer to extract a target-aware representation: $h^a_{att} = \sum_{t=1}^T a_t h^a_{pool,t}$, with $a_t$ computed by the local activation unit [Zhou et al., 2018b]. The concatenation $h^a = [h^a_{pool}, h^a_{L_b,T}, h^a_{att}]$ is served as the sequential feature extracted from Stage 1. In practice, this stage remains the procedure to execute looking up embedding only once, which is orthogonal to the industrial large-scale sparse-features lookup optimizations [Oldridge et al., 2020].

Stage 2: Interaction Exploration on Non-sequential Features. Learning effective feature interactions is crucial for recommender systems. Different from the widely-used methods that manually extract useful interactions or exhaustively enumerate interactions of bounded degrees [Cheng et al., 2016; Guo et al., 2017; Lian et al., 2018], AMEIR explores a small number of $M$ beneficial interactions $p_i = \{p^i_1, p^i_2, ..., p^i_{M_i}\}$ from the search subspace of interactions. The search subspace contains all low-order and high-order interactions among the non-sequential features, where each interaction has the same dimension as the input embeddings, i.e., $p_i^j \in \mathbb{R}^K$, $i \in [1, 2, ..., M]$. 

Stage 3: Aggregation MLP Investigation with Shortcut Connection. Given the inputs of sequential features, non-sequential features, and explored interactions $h^c = [h^c, e^c, p^c]$, AMEIR jointly learns an MLP with dense Squeeze-and-Excitation (SE) [Hu et al., 2018] connection to aggregate these features by taking advantages of memorization from wide shortcut connection and generalization from deep neural network. Assume the aggregation has $L_c$ hidden layers. Considering both recommendation quality and efficiency, AMEIR searches for the dimension and activation function for each hidden layer from the search subspace of MLP. The forward pass of each hidden layer $l \in [1, 2, ..., L_c]$ can be depicted as:

$$h^l = \text{Act}^c_l(W^c_l h^{c-1}_l + b^c_l),$$

where $h^c_l$, $W^c_l$, $b^c_l$ and $\text{Act}^c_l$ are the output, weight, bias and activation function of the $l$-th layer. Meanwhile, inspired by the bypaths in [Cheng et al., 2016; Guo et al., 2017; Lian et al., 2018], AMEIR introduces a dense connection to help memorize the low-order information from the feature embeddings and explored interactions $Q^{se} = [e^c, p^c] = [e^c_1, e^c_2, ..., e^c_T, p^c_1, ..., p^c_M]$. As the dimension of $Q^{se}$ is several times higher than $h^c_{L_c}$, AMEIR employs the SE shortcut connection with 1.0 expansion ratio for attentive recalibration and feature compression [Liu et al., 2020]. The attention units $a^{se} = [a^{se}_1, a^{se}_2, ..., a^{se}_{M+N}]$ in the SE network are computed by:

$$a^{se} = \sigma(W^{se}_2 \text{ReLU}(W^{se}_1 Q^{se} v^{se})), $$

where $v^{se}$ is applied for linear projection, and $W^{se}_1, W^{se}_2$ are weight matrices in the fully-connected layers. The output of SE connection is given by the weighted sum $h^{se} = \sum_{i=1}^{M+N} a^{se}_i q^{se}_i$, where $q^{se}_i$ is the $i$-th element in $Q^{se}$. AMEIR concatenates the results from aggregation MLP and SE connection as the output representation $h^e = [h^c_{L_c}, h^{se}]$.

Loss Function. For the CTR prediction task, AMEIR mini-
mizes the binary log loss for optimization:

\[
L = -\frac{1}{|S|} \sum_{(x,y) \in S} \left[ y \log \sigma(w^T h^c) + (1-y) \log(1-\sigma(w^T h^c)) \right],
\]

where \(\sigma(w^T h^c)\) is the predicted CTR score, and \(S\) is the training set with \(x\) as input and \(y \in \{0, 1\}\) as label. For the item retrieval task, AMEIR adopts the sampled binary cross entropy loss as the objective function:

\[
L = -\frac{1}{|S|} \sum_{(x,i) \in S} \left[ \log \sigma(e_i^T Wh^c) + \sum_{j \in \mathcal{I}_c} \log(1-\sigma(e_j^T Wh^c)) \right],
\]

where \(\sigma(e_i^T Wh^c)\) is the relevance score between target item \(i\) and final representation; \(S\) is the training set of input feature \(x\) and target item \(i\); \(\mathcal{I}_c\) is the item set of negative samples.

### 2.2 Search Subspaces of Three Stages

For each stage in the backbone, AMEIR introduces a corresponding search subspace to cover the representative recommendation models. We will describe each of the three subspaces below and show a detailed comparison in Sec.2.4.

#### Search Subspace of Building Blocks

In order to cover the existing behavior modeling methods, AMEIR proposes a search subspace of building blocks in the behavior modeling network, including three operation sets for normalization, layer, and activation. Specifically, AMEIR collects the normalization set of \{Layer normalization [Ba et al., 2016], None\} and activation set of \{ReLU, GeLU [Vaswani et al., 2017], Swish [Ramachandran et al., 2017], Identity\} that are commonly used in the previous methods. Regarding the layer operations, AMEIR introduces four categories of candidate layers, namely convolutional layers, recurrent layers, pooling layers, and attention layers to identify sequential patterns in the user history. The convolutional layers are all one-dimension, including standard convolution with kernel size \{1, 3\} and dilated convolution with kernel size \{3, 5, 7\}. The pooling layers consist of average pooling and max pooling with kernel size 3. Both convolutional and pooling layers are of stride 1 and SAME padding. Bi-directional GRU (Bi-GRU) is employed as the recurrent layer as it is faster than Bi-LSTM without loss of precision. Two-head and four-head bi-directional self attentions [Vaswani et al., 2017] are also introduced for better behavior modeling. Additionally, when the target item appears in the input, the layer operation set will associate an attention layer that attends to each sequence position from target [Zhou et al., 2018b]. Besides, zero operation is also included to implicitly allow a dynamic depth of behavior modeling network by dropping redundant layers. It is worth emphasizing that the operation sets are proposed to cover the existing methods. When a new operation is developed, it can be easily added to the corresponding operation set to help find a better recommendation model.

#### Search Subspace of Interactions

The existing literature adopts Hadamard product [Lian et al., 2018], inner product [Guo et al., 2017; Qu et al., 2018], bilinear function [Huang et al., 2019] and cross product [Cheng et al., 2016] in the feature interactions. Hadamard product calculates the element-wise product among the feature embeddings, which can be exploited by both MLP and shortcut connection. The inner product can be seen as a simple form of Hadamard product compressed by sum pooling. Bilinear function extends Hadamard product by introducing extra parameters to learn fine-grained representations, but it can not handle high-order interactions and does not exhibit superiority over Hadamard product when combining with MLP [Huang et al., 2019]. Although the cross product can yield a more flexible form of interactions, it can only be used in the shortcuts [Luo et al., 2019] because the parameters will exponentially explode in the order of interactions and can not be well trained due to the low frequency of occurrence. Therefore, AMEIR chooses Hadamard product as the interaction function due to its expressive power and flexibility. The search subspace of interactions includes all low-order and high-order Hadamard interactions. An interaction of order \(r\) can be expressed as \(e_{i_1}^b \circ e_{i_2}^b \circ \ldots \circ e_{i_r}^b\) where \(\circ\) refers to the Hadamard product and \(e_{i_1}^b, e_{i_2}^b, \ldots, e_{i_r}^b\) are selected from \(e^b\).

### 2.3 Three-step One-shot Searching Pipeline

One-shot NAS in AMEIR. Recent approaches [Bender et al., 2018; Pham et al., 2018] introduce efficient one-shot weight-sharing paradigm in NAS to boost search efficiency, where all child models share the weights of common operations in a large one-shot model that subsamples every possible architecture in the search space. Though diverse complex search algorithms [Cai et al., 2019b; Chen et al., 2019b; Chu et al., 2019; Dong and Yang, 2019b] have been coupled with one-shot NAS, the latest studies [Bender et al., 2018; Dong and Yang, 2020; Li and Talwalkar, 2019] reveal that the efficient one-shot random search is surprisingly better than the complex gradient-based methods of DARTS [Liu et al., 2019], SNAS [Xie et al., 2019] and RL-based methods of ENAS [Pham et al., 2018]. To facilitate the industrial demands of agile development and architecture iteration, AMEIR realizes the one-shot random search in the recommendation model: (1) The child models are randomly sampled from the search spaces (actually subspaces) to train the shared weights of the one-shot model and validate its performance on a small subset of validation instances to ensure adequate training while avoiding overfitting. (2) A subset of child models is randomly selected and evaluated on the same validation subset using the inherited weights from the one-shot model. (3) The child models with the best performance will be derived and retrained for the final recommendation outcome. Moreover, to reduce the magnitude of the search space and find the better recommendation models, AMEIR modularizes the search process into three steps matching the three stages in the backbone model by progressively searching for the architectures in the three search subspaces [Li et al., 2019;
Sciuto et al., 2020]. Meanwhile, AMEIR ensures the fairness [Chu et al., 2019] of one-shot model training to alleviate the representation shift caused by weight sharing [Chu et al., 2019; Yu et al., 2020b], where all candidates are uniformly sampled and activated per gradient update. Details of the three steps are shown below.

**Step 1: Behavior Modeling Search.** In Step 1, AMEIR searches for the behavior modeling networks from the search subspace of building blocks, as shown in Fig.1. The normalization, layer, and activation operations are randomly selected from the corresponding operation sets to construct the recommendation model. To decouple behavior modeling from other stages, a pre-defined MLP is exploited to combine $h^a$ and $e^b$ during Step 1. The training and evaluation procedures are the same as previously described.

**Step 2: Interaction Exploration.** In Step 2, AMEIR combines one-shot NAS with the sequential model-based optimization (SMBO) algorithm [Liu et al., 2018a] to explore interactions on non-sequential features. The interaction set $p^b$ is initialized with non-sequential features $e^b$, and then progressively updated by several cycles of evolution to increase the interaction order, as illustrated in Fig.1: (a) The candidate interactions in $p^b$ intersect with all non-sequential fields from $e^b$ via Hadamard product to mutate for higher-order interactions. The expanded interaction set is denoted by $p^b'$. (b) The intersections in $p^b'$ are trained by one-shot NAS and evaluated on the validation set. (c) Top-$k$ interactions with the highest validation fitness are retained as the new $p^b$, while other interactions are filtered out to avoid exhaustively visiting all possible solutions. After the evolution, AMEIR selects top-$M$ features in $p^b$ as the resulting interaction set.

In order to train and evaluate interactions in the one-shot model, AMEIR utilizes another pre-defined MLP to aggregate interactions in $p^b$ with non-sequential embeddings in $e^b$, while the behavior modeling here is left out for efficiency. As the linear projection of feature concatenation is equivalent to feature-wise addition of projected results, i.e., $Wh^T = [W_1, W_2, ..., W_N][h_1, h_2, ..., h_N]^T = \sum_{i=1}^{N} W_i h_i^T$, the one-shotNAS in Step 2 is efficiently realized by adding the projections of one uniformly-sampled interaction and all embeddings in $e^b$ as the output of the first hidden layer, without accessing to all interactions in $p^b$. To further speed up the search process, the weights of the one-shot model are inherited in consecutive evolution cycles.

**Step 3: MLP Investigation.** In Step 3, AMEIR investigates the dimensions and activation functions in the aggregation MLP based on the behavior modeling network and interactions found in the previous steps. To find the hidden sizes of MLP, AMEIR pre-allots a weight matrix with the maximum dimension of $(K^c_{0}, K^c_{0})$, i.e., $W^c \in \mathbb{R}^{K_0 \times K_0}$, for each hidden layer [Guo et al., 2019]. Suppose $h_{in}$ and $h_{out}$ are chosen to be the hidden sizes of previous and current layers. AMEIR will slice out sub-matrix $W^c[: h_{in} :, h_{out}]$ to assemble the MLP in Eqn. (4). Other training and evaluation procedures are similar to Steps 1 and 2.

**Derivation.** After the three-step random search, AMEIR assembles the search results as the final outcome and retrains the derived model for evaluation. It is worth emphasizing that the derived model is initialized with the one-shot model weights because the training protocols of model architectures and datasets are the same for both one-shot model and child models, which is in contrast to the most existing methods that adopt proxy task for one-shot model training [Liu et al., 2019; So et al., 2019; Xie et al., 2019; Yu et al., 2020b]. The inherited weights accelerate the retraining process and compensate for the search cost.

**2.4 Related Works to AMEIR**

**Behavior Modeling.** The existing methods introduce RNN [Hidasi and Karatzoglou, 2018], CNN [Yuan et al., 2019], or Transformer-based [Sun et al., 2019] behavior modeling networks to capture sequential relations and evolved interests from user behaviors. Then, the sequential features are extracted from the sequence-level hidden states by either taking the last hidden state [Kang and McAuley, 2018] or aggregating the sequence by a pooling layer [Cheng et al., 2016; Covington et al., 2016]. Some recent works [Zhou et al., 2018b; Zhou et al., 2019] further employ an attention mechanism between the sequential representations and target item to retrieve the user’s main proposal. These methods can all be found in AMEIR’s search space. Specifically, the search subspace of building blocks contains all layer, activation, and normalization operations in those behavior modeling networks. Moreover, the last hidden state, pooling-based aggregation, and attention-based aggregation are all used for sequence compression.

**Interaction Exploration.** In parallel with behavior modeling, some works resort to explore feature interactions in the DL-based recommendation models. LR [Cheng et al., 2016; Luo et al., 2019], FM [Guo et al., 2017; Liu et al., 2020], DCN [Wang et al., 2017] and xDeepFM [Lian et al., 2018] are exploited to import explicit low-order or high-order interactions through shortcut connections, i.e., concatenating interactions to the last hidden layer of MLP, and then apply linear projection with learnable variables or all-one vectors to generate the prediction scores jointly with the hidden layers. Other than the shortcut connection, FibiNet [Huang et al., 2019] take interactions as the input of MLP, while PNN [Qu et al., 2016; Qu et al., 2018] further combines interactions with raw embeddings to induce more features in the network. It is not hard to find that all these methods can be derived from AMEIR’s search space. Concretely, all low-order and high-order interactions can be extracted from the search subspace of interactions. Meanwhile, raw input embeddings and explored interactions are used by both shortcut connection and MLP, covering the existing interaction-combining strategies. Besides, the attentive SE connection could implicitly characterize attention-based, regression-based, and FM-based shortcut connections.

**MLP Investigation.** The existing literature [Covington et al., 2016; Guo et al., 2017; Zhou et al., 2018b] mainly uses sub-optimal hand-crafted MLPs to aggregate features, where activation functions and hidden sizes are designed manually or by grid search. However, the search space of MLP is large and these hand-crafted MLP settings are always sub-optimal. AMEIR introduces a discrete yet flexible search space that covers most of the common MLP settings and finds the MLP
settings through the reliable and learnable neural architecture search.

Based on the above analysis, it can be inferred that AMEIR’s search space covers most of the representative recommendation models. Therefore, AMEIR can perform as good as these methods in various scenarios, and further derive better models when conducting sufficient training and exhaustive search. A more detailed comparison between AMEIR and the related works is given in Appendix A.

3 Experiments

In Sec.3, we will compare AMEIR with competitive baselines on various recommendation scenarios, study the impact of different components, and conduct the efficiency analysis on the proposed method. We will also describe more details of the experiments in Appendix B-E, including datasets, baseline methods, implementations, and search results.

3.1 Industrial Results for AMEIR

To verify all three steps of AMEIR in the industrial scenario, we refer to the Alimama CTR dataset [Feng et al., 2019] that comprises both sequential and non-sequential features for a comprehensive comparison. We use similar preprocessing settings as [Feng et al., 2019], where the embedding size is set to 4 and the maximum sequence length is set to 50. The model, and the pre-defined MLP is set to [200, 80] during Steps 1 and 2. By conforming to [Li and Talwalkar, 2019; Liu et al., 2019], we run AMEIR for 4 times and report the mean metrics across 4 runs as the final results.

Tab. 1 depicts the results on Alimama dataset. AMEIR-A (only uses behavior modeling result from Step 1) presents competitive performance with the promising empirical architectures DIEN and DSIN. Furthermore, a significant improvement (note that an improvement of AUC around 0.0005-0.001 is already regarded as practically significant in the industrial scenarios [Feng et al., 2019]) can be observed when combining AMEIR-A with searched interactions (AMEIR-B) and MLP (AMEIR-C). An interesting point on the Alimama dataset is that compared to sequential modeling and interaction exploration, the MLP search significantly contributes to the final results, which indicates that all three stages/steps in AMEIR are necessary for high-quality recommendations.

3.2 Applicable for Various Scenarios

To show AMEIR is applicable for various recommendation tasks, we evaluate AMEIR on two additional scenarios, namely sequential scenario, and non-sequential scenario: (1) The sequential scenario only presents sequential user behaviors to examine Step 1 behavior modeling. The training is based on the item retrieval task. The sequential datasets include Amazon Beauty and Steam [Kang and McAuley, 2018]. (2) The non-sequential scenario is introduced to verify the Step 2 interaction exploration and Step 3 MLP investigation, where only non-sequential features are provided for the recommendation. The non-sequential datasets include benchmarking CTR datasets Criteo [Guo et al., 2017] and Avazu [Qu et al., 2018]. The implementation details are similar to the industrial scenario and can be found in Appendix D.

Evaluation on Sequential Scenario. Tab. 2 summarizes the results on Beauty and Steam datasets. The hand-crafted methods, including the strongest baselines BERT4Rec and NextItNet, can not dominate both datasets, which empirically justifies that the best architecture is usually task-aware. In contrast, AMEIR consistently achieves better accuracy compared to the baselines with well-designed architectures, indicating the strength of AMEIR and the capability of searching better models from the search subspace of building blocks.

Evaluation on Non-sequential Scenario. Tab. 3 presents the results on non-sequential datasets. It can be observed that AMEIR-I (DeepFM backbone with interactions explored by AMEIR) consistently surpasses competitive hand-crafted methods.
baselines xDeepFM and FibiNet as well as AutoML-based baselines AutoCross and AutoFIS on different tasks over multiple runs, validating the efficacy of interaction exploration in Step 2. Moreover, our method gains remarkable improvement over AMEIR-I and exceeds AutoCTR when combining explored interactions with the SE connection (AMEIR-B) and MLP investigation (AMEIR-C). The result verifies the design of both the backbone and search subspaces.

### 3.3 Ablation Study

#### Ablation Study on NAS Methods.
To study the importance of the proposed search method, we evaluate the performance of different NAS algorithms on the same search space of the Alimama dataset. As shown in Tab. 4, the one-shot random search achieves better performance than both brute-force methods of random, RL, EA, and the algorithmic complex methods of ENAS, DARTS, SNAS with lower search cost. Moreover, AMEIR with the three-stage one-shot searching pipeline brings about a remarkable improvement on the one-shot random search under a similar time budget, demonstrating the effectiveness and efficiency of the proposed method.

#### Ablation Study on Backbone Model.
To further analyze the impact of the backbone model, we isolate different components in Stages 2 and 3, and report the mean metrics on the non-sequential datasets. As shown in Tab. 5, the interactions (searched) applied to MLP and FM bring orthogonal improvement on vanilla DNN. The result reveals that both combining strategies contribute to the result. Additionally, SE interaction (AMEIR-B) and MLP investigation (AMEIR-C) could further boost the performance of recommendation, suggesting the superiority of the backbone design in AMEIR.

### 3.4 Efficiency in Search and Inference Phases

#### Search Efficiency.
The search cost of AMEIR on the Beauty, Steam, Avazu, Criteo, Alimama datasets are 0.2, 0.2, 0.25, 0.25, 0.5 GPU-days respectively on a single Tesla V100 GPU, which are all similar to training the individual models of baseline methods. As AMEIR uses the trained one-shot weights to initialize the derived model, the fine-tuning time could generally be ignored. Therefore, the overall time cost of AMEIR is roughly the same as the methods that do not employ NAS. The result proves the efficiency of AMEIR in the search phase, and it is evident that AMEIR can help facilitate the design of recommender systems in real-world scenarios.

#### Inference Efficiency.
We record the queries/samples-per-second (QPS) of AMEIR and the compared models to examine the inference speed on the Alimama dataset in Tab. 4. Apart from the improvement in accuracy, AMEIR is also competitive in efficiency compared with the commonly-used attention-based models. Besides, the searched MLPs are always much smaller than the hand-crafted ones (details shown in Appendix E), such that the numbers of parameters of the searched models are less than half of the baselines. The results indicate that instead of increasing the model complexity, AMEIR coordinates the three parts of a recommender system to find the architecture with the best practice.

### 4 Conclusion

In this paper, we introduce AMEIR for automatic behavior modeling, interaction exploration, and MLP investigation in the recommender systems to relieve the manual efforts of feature engineering and architecture engineering. Owing to the three-stage search space and the matched three-step searching pipeline, AMEIR covers most of the representative recommendation models and outperforms the competitive baselines in various recommendation scenarios with lower model complexity and comparable time cost, demonstrating efficiency, effectiveness, and robustness of the proposed method.

### References

[Ba et al., 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Belletti et al., 2019] Francois Belletti, Minmin Chen, and Ed H Chi. Quantifying long range dependence in language and user behavior to improve rnns. In *SIGKDD*, 2019.

[Bender et al., 2018] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, pages 550–559, 2018.

[Bergstra and Bengio, 2012] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13(1):281–305, 2012.

[Cai et al., 2019a] Han Cai, Chuan Gen, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2019.

[Cai et al., 2019b] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[Chen et al., 2019a] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st International Workshop on Deep Learning*
[Chen et al., 2019b] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Deta: Backbone search for object detection. In NeurIPS, pages 6638–6648, 2019.

[Cheng et al., 2016] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Harsh Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In DLRS, pages 7–10, 2016.

[Chu et al., 2019] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv preprint arXiv:1907.01845, 2019.

[Covington et al., 2016] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In RecSys, 2016.

[Devlin et al., 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL, pages 4171–4186, 2019.

[Dong and Yang, 2019a] Xuanyi Dong and Yi Yang. Nass-bench-201: Extending the scope of reproducible neural architecture search. In International Conference on Learning Representations, 2019.

[Dong and Yang, 2019b] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In NeurIPS, pages 759–770, 2019.

[Dong and Yang, 2020] Xuanyi Dong and Yi Yang. Nass-bench-201: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326, 2020.

[Fan et al., 2019] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In WWW, pages 417–426, 2019.

[Feng et al., 2019] Yufei Feng, Fuyu Lv, Weichen Shen, Menghan Wang, Fei Sun, Yu Zhu, and Keping Yang. Deep session network for click-through rate prediction. In AAAI, pages 2301–2307. AAAI Press, 2019.

[Guo et al., 2017] Huifeng Guo, Ruiming Tang, Yuming Ye, Zhenguo Li, and Xiupiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In IJCAI, pages 1725–1731, 2017.

[Guo et al., 2019] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420, 2019.

[He and Chua, 2017] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In SIGIR, pages 355–364, 2017.

[He et al., 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In ICCV, 2016.

[He et al., 2017a] Ruining He, Wang Cheng Kang, and Justin Mcauley. Translation-based recommendation. In RecSys, 2017.

[He et al., 2017b] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In WWW, pages 173–182, 2017.

[Hidasi and Karatzoglou, 2018] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In CIKM, pages 843–852. ACM, 2018.

[Hidasi et al., 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and D Tikk. Session-based recommendations with recurrent neural networks. In ICLR, 2016.

[Hu et al., 2018] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, 2018.

[Huang et al., 2019] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. Fibenet: combining feature importance and bilinear feature interaction for click-through rate prediction. In RecSys, pages 169–177, 2019.

[Jang et al., 2017] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In ICLR, 2017.

[Kang and McAuley, 2018] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In ICDM. IEEE, 2018.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[Li and Talwalkar, 2019] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In UAI, 2019.

[Li et al., 2017] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In CIKM, 2017.

[Li et al., 2019] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. arXiv preprint arXiv:1910.02543, 2019.

[Lian et al., 2018] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhengxia Chen, Xing Xie, and Guangzhong Sun. Xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In KDD, 2018.

[Liu et al., 2018a] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In ECCV, 2018.

[Liu et al., 2018b] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: short-term attention/memory priority model for session-based recommendation. In KDD, pages 1831–1839, 2018.

[Liu et al., 2019] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In ICLR, 2019.
[Liu et al., 2020] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiujian He, Zhen-guo Li, and Yong Yu. Autofts: Automatic feature interaction selection in factorization models for click-through rate prediction. In *KDD*, 2020.

[Luo et al., 2019] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *KDD*, pages 1936–1945, 2019.

[McAuley et al., 2015] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

[Mnih and Salakhutdinov, 2008] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *NeurIPS*, pages 1257–1264, 2008.

[NVIDIA, 2020] NVIDIA. Hugectr: a high-efficiency gpu framework for click-through-rate (ctr) estimation training. 2020.

[Oldridge et al., 2020] Even Oldridge, Julio Perez, Ben Frederickson, Nicolas Koumchatzky, Minseok Lee, Zehuan Wang, Lei Wu, Fan Yu, Rick Zamora, Onur Yilmaz, et al. Merlin: A gpu accelerated recommendation framework. 2020.

[Pham et al., 2018] Hieu Pham, melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

[Qu et al., 2016] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. In *ICDM*, pages 1149–1154. IEEE, 2016.

[Qu et al., 2018] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiujian He. Product-based neural networks for user response prediction over multi-field categorical data. *TOIS*, 2018.

[Quadrana et al., 2017] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*, pages 130–137, 2017.

[Radosavovic et al., 2020] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[Ramachandran et al., 2017] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[Real et al., 2017] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, pages 2902–2911. JMLR. org, 2017.

[Real et al., 2019] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[Rendle et al., 2009] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461. AUAI Press, 2009.

[Rendle et al., 2010] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pages 811–820, 2010.

[Rendle, 2010] Steffen Rendle. Factorization machines. In *ICDM*, pages 995–1000. IEEE, 2010.

[Saxe et al., 2014] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014.

[Schulman et al., 2017] John Schulman, Filip Wolski, Pra-fulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Scuito et al., 2020] Christian Scuito, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020.

[So et al., 2019] David So, Quoc Le, and Chen Liang. The evolved transformer. In *ICML*, pages 5877–5886, 2019.

[Song et al., 2020] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. Towards automated neural interaction discovery for click-through rate prediction. In *KDD*, pages 945–955, 2020.

[Sun et al., 2019] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*, 2019.

[Tan and Le, 2019] Mingxing Tan and Quoc Le. Efficientnet: Rethinking network scaling for convolutional neural networks. In *ICML*, 2019.

[Tang and Wang, 2018] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*, pages 565–573, 2018.

[Vaswani et al., 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Wang et al., 2017] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD’17*, 2017.

[Wang et al., 2020a] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenes: Improving neural architecture search via attentive sampling. *arXiv preprint arXiv:2011.09011*, 2020.
[Wang et al., 2020b] Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. Textnas: A neural architecture search space tailored for text representation. In AAAI, 2020.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992.

[Wu et al., 2017] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In WSDM, 2017.

[Wu et al., 2019a] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In CVPR, pages 10734–10742, 2019.

[Wu et al., 2019b] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In AAAI, volume 33, pages 346–353, 2019.

[Xia et al., 2017] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: learning the weight of feature interactions via attention networks. In IJCAI, pages 3119–3125, 2017.

[Xie et al., 2019] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In ICLR, 2019.

[Xu et al., 2019] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. Graph contextualized self-attention network for session-based recommendation. In IJCAI, pages 3940–3946, 2019.

[Ying et al., 2018] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In KDD, pages 974–983, 2018.

[Yu et al., 2020a] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In ECCV, 2020.

[Yu et al., 2020b] Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. How to train your super-net: An analysis of training heuristics in weight-sharing nas. arXiv preprint arXiv:2003.04276, 2020.

[Yuan et al., 2019] Fujie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In WSDM, pages 582–590, 2019.

[Zhang et al., 2020] Yuanxing Zhang, Pengyu Zhao, Yushuo Guan, Lin Chen, Kaigui Bian, Lingyang Song, Bin Cui, and Xiaoming Li. Preference-aware mask for session-based recommendation with bidirectional transformer. In ICASSP, pages 3412–3416. IEEE, 2020.

[Zhou et al., 2018a] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. Atrank: An attention-based user behavior modeling framework for recommendation. In AAAI, 2018.

[Zhou et al., 2018b] Guorui Zhou, Xiaoliang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In KDD, pages 1059–1068, 2018.

[Zhou et al., 2019] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In AAAI, pages 5941–5948, 2019.

[Zoph and Le, 2017] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In ICLR, 2017.

[Zoph et al., 2018] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In CVPR, 2018.
Appendix

A Detailed Comparison between AMEIR and Representative Methods

In Appendix A, we conduct a detailed comparison between AMEIR and the representative recommender systems on behavior modeling and interaction exploration to show that these methods are all covered by AMEIR’s search space.

A.1 Behavior Modeling

The existing behavior modeling methods can be categorized into RNN-based, CNN-based, Transformer-based methods, and mixed methods.

• RNN-based methods [Hidasi et al., 2016; Hidasi and Karatzoglou, 2018; Quadrana et al., 2017; Wu et al., 2017] mainly apply LSTM or GRU layers to extract sequential representations, which can be covered by the building blocks with recurrent layer operations.

• CNN-based methods apply convolutional layers on the sequential representations. Caser [Tang and Wang, 2018] introduces horizontal and vertical convolutions by sliding windows on the sequential embedding and applies max pooling over the extracted feature maps for compression. However, sequence-level pooling on one hidden convolution layer is hard to capture complex relations in the sequence. NextItNet [Yuan et al., 2019] utilizes stacked residual blocks of dilated convolution, which can be roughly regarded as a special case of the behavior modeling network in AMEIR because both dilated convolutional block and standard $1 \times 1$ convolutional block are included in the search subspace of Stage 1.

• Transformer-based methods, e.g., ATRank [Zhou et al., 2018a], SASRec [Kang and McAuley, 2018], and BERT4Rec [Sun et al., 2019] introduce self-attention layers and point-wise feed-forward layers to capture long-term semantics, which are the same as stacking multi-head attention blocks and $1 \times 1$ convolutional blocks in the behavior modeling network.

• Mixed methods such as DIEN [Zhou et al., 2019] and DSIN [Feng et al., 2019] employ the mixture of the above-mentioned layers, which potentially take advantages of these operations from various aspects. DIEN introduces GRU + AUGRU to capture interest evolution in user behavior. DSIN employs self-attention + Bi-LSTM to model inner- and intra-relationship in multiple historical sessions. Both DIEN and DSIN (if neglecting the session division) can be approximated by sequences of building blocks in AMEIR with the specified layer operations, i.e., \{Bi-GRU, Attention from Target, Bi-GRU\} and \{4-Head Self Attention, $1 \times 1$ Convolution, Bi-GRU\} respectively.

On top of the behavior modeling, these methods either extract the last hidden state [Hidasi et al., 2016; Kang and McAuley, 2018], or employ pooling layer [Tang and Wang, 2018] and attention layer from target [Zhou et al., 2018b; Zhou et al., 2019] to compress the sequential hidden states into a fixed-length representation, which are all comprised by the behavior modeling module of Stage 1 in AMEIR.

A.2 Interaction Exploration

In Sec.2, we have declared that AMEIR can cover all feature interactions and interaction-combining strategies in the representative interaction exploration methods. For a more formal and intuitive comparison, we develop the computation process of final prediction scores in both AMEIR and the representative recommender systems and show that all compared methods can be explicitly or implicitly included in AMEIR’s search space. Following the existing symbol usage, $x^b$ and $e^b$ denote the non-sequential input features and corresponding field embeddings. For the convenience of representation, we use $p^b$ to represent both investigated interactions and enumerated bounded-degree interactions of various types, including cross-product interactions, bilinear interactions, and Hadamard-product interactions (inner-product interaction can be expressed as a special form of Hadamard-product interaction, i.e., $1^T p^b_i$, where $p^b_i$ represents a Hadamard-product interaction) according to the related literature. We use $a$ to denote the attention weights of interactions and embeddings computed by compressed interaction network (CIN) [Lian et al., 2018] or attention module [Liu et al., 2020; Xiao et al., 2017]. The bias in the MLP layers is omitted for convenience. We compare AMEIR with representative methods of LR, FM [Rendle, 2010], AFM [Xiao et al., 2017], DNN [Covington et al., 2016], Wide & Deep [Cheng et al., 2016], AutoCross [Luo et al., 2019] (Wide & Deep version), NFM [He and Chua, 2017], DeepFM [Guo et al., 2017], DCN [Wang et al., 2017], xDeepFM [Lian et al., 2018], AutoFIS [Liu et al., 2020], PNN [Qu et al., 2018] and FibitNet [Huang et al., 2019].
(SE feature interactions implicitly included in p):

\[
\begin{align*}
    y_{LR} &= w_x^T x^b \\
    y_{FM} &= 1^T \sum_i p_i^b + w_{x}^T x^b \\
    y_{AFM} &= w_p^T (\sum_i a_i p_i^b) + w_{x}^T x^b \\
    y_{DNN} &= w_{MLP}(e^b) \\
    y_{Wide & Deep} &= w_{MLP}(e^b) + w_x^T p^b + w_{x}^T x^b \\
    y_{AutoCross} &= w_{MLP}(e^b) + w_x^T p^b + w_{x}^T x^b \\
    y_{NFM} &= w_{MLP}(e^b, \sum_i p_i^b) + w_{x}^T x^b \\
    y_{DeepFM} &= w_{MLP}(e^b) + 1^T \sum_i p_i^b + w_{x}^T x^b \\
    y_{DCN} &= w_{MLP}(e^b) + w_x^T (\sum_i a_i p_i^b) + w_{e}^T (\sum_j e_j^b) \\
    y_{DeepDCN} &= w_{MLP}(e^b) + w_x^T (\sum_i a_i p_i^b) + w_{x}^T x^b \\
    y_{AutoFIS} &= w_{MLP}(e^b) + 1^T (\sum_i a_i p_i^b) + w_{x}^T x^b \\
    y_{FISNet} &= w_{MLP}(p^b) + w_{x}^T x^b \\
    y_{PNN} &= w_{MLP}(e^b, p^b) \\
    y_{AMEIR} &= w_{MLP}(e^b, p^b) + w_p^T (\sum_i a_i p_i^b) + w_e^T (\sum_j e_j^b),
\end{align*}
\]

where \( w_x, w_e, w_p \) and \( w_M \) are the weights of linear projections on the input features, input embeddings, feature interactions, and final hidden state of MLP. It can be inferred that the MLP modules \( MLP(e^b) \) (MLP on embedding), \( MLP(p^b) \) (MLP on interactions) and \( MLP(e^b, \sum_i p_i^b) \) (MLP on embedding and compressed interactions) are all special cases of \( MLP(e^b, p^b) \) (MLP on embedding and interactions), while the interaction modules \( 1^T \sum_i p_i^b \) (FM on interactions) and \( 1^T (\sum_i a_i p_i^b) \) (FM on weighted interactions) are all comprised by SE connection \( w_x^T (\sum_i a_i p_i^b) \) (regression on weighted interactions). Besides, the bypass \( w_e^T (\sum_j e_j^b) \) can implicitly capture the semantics of linear regression on the raw inputs of \( w_x^T x^b \) and cross network on the input embedding of \( w_e^T (\sum_j e_j^b) \). Therefore, all compared methods are explicitly or implicitly contained in AMEIR’s search space.

## B Datasets

In Appendix B, we introduce the datasets in the industrial scenario, sequential scenario, and non-sequential scenario. Tables 6-8 show the statistics of the datasets.

### B.1 Industrial Scenario

Table 6: Statistics of Alimama dataset in the industrial scenario.

| # Users | # Sparse Feature | # Dense Feature | # (cate, id, brand) | Avg. Seq. Length | # Train Traces | # Test Traces | Prop. of Click |
|---------|-----------------|----------------|---------------------|-----------------|---------------|--------------|---------------|
| 265,443 | 15              | 1              | 1,027,443           | 46.31           | 5,544,213     | 660,694      | 0.0514        |

The Alimama dataset in the industrial scenario contains data of all formats mentioned in Sec. 3 so that it is used for comprehensive comparison in all three-stage pipeline of AMEIR: Alimama provides 26 million records of ad display/click logs in 8 days, where for each user it keeps track of 200 recent shopping behaviors in 22 days. The logs in the first 7 days are used for training and the logs in the last day are treated as the test set. Each data instance has 15 sparse features, including user profile,

1https://tianchi.aliyun.com/dataset/dataDetail?dataId=56
user’s sequential behaviors, and context profile. The dimension of these features ranges from 3 to 309,449. There is only 1 dense feature of "price" which has been normalized to [-1,1]. We randomly sample 25% of the dataset and remains 6,204,907 data instances following [Feng et al., 2019]. 5% of the training set is used for validation in the architecture search and model selection. As there are many users that do not have any records of sequential behaviors, we assign a special tag to the behavior sequences of these users to avoid the behavior modeling network executing on vacant inputs.

### B.2 Sequential Scenario

#### Table 7: Statistics of Beauty and Steam datasets in the sequential scenario.

| # User | # Item | Min. Length | Max. Length | Avg. Length | Density |
|--------|--------|-------------|-------------|-------------|---------|
| Beauty | 22,363 | 12,101      | 5           | 204         | 8.876   | 0.07%   |
| Steam  | 281,210| 11,961      | 5           | 1,226       | 12.391  | 0.10%   |

The Amazon Beauty and Steam datasets in the sequential scenario are proposed to evaluate the Stage 1 behavior modeling in AMEIR, which only include sequential behavior of each user, and the training is based on next-item prediction with Eqn. (7):

- Amazon[^2] is composed of reviews and metadata from Amazon.com [McAuley et al., 2015], where data instances are categorized into separate datasets according to the product category. In this paper, we use the category of “Beauty” for evaluation.
- Steam[^3] is introduced by [Kang and McAuley, 2018] which contains millions of review information from the Steam video game platform.

Following the common practice [He et al., 2017b; Kang and McAuley, 2018], the reviews in the datasets are treated as implicit feedback. We group data instances by users and sort them according to the timestamps. The users and items with less than 5 feedbacks are removed from datasets. The leave-one-out strategy is adopted for partitioning [He et al., 2017b; Kang and McAuley, 2018; Sun et al., 2019], where for each user the last two actions are treated as the validation set and test set, and the remaining actions are used for training.

### B.3 Non-sequential Scenario

#### Table 8: Statistics of Criteo and Avazu datasets in the non-sequential scenario.

| # Train Instances | # Test Instances | Fields | Prop. of Click |
|-------------------|------------------|--------|----------------|
| Criteo            | 41,256,555       | 4,584,062 | 39            | 25.62%        |
| Avazu             | 32,343,173       | 8,085,794 | 24            | 16.98%        |

The Criteo and Avazu datasets in the non-sequential scenario only include non-sequential categorical features for CTR prediction to verify the Stage 2 interaction exploration and Stage 3 aggregation MLP investigation:

- Criteo[^4] contains 7 days of click-through data, which is widely used for CTR prediction benchmarking. There are 26 anonymous categorical fields and 13 continuous fields in the Criteo dataset. The continuous fields are normalized to [-1,1] and only used for MLP input while categorical fields are applied to both MLP and shortcut connection as well as the interaction exploration module in AMEIR. The dataset is randomly split into two parts while maintaining the label distribution: 90% is for training and the rest is for testing. Besides, 10% of the training set is used for validation.
- Avazu[^5] consists of 11 days of online ad click-through data. Each data instance includes 24 categorical fields that represent profiles of a single ad impression. We randomly split the dataset into two parts, where 80% is for training and the rest is for testing. Same as the Criteo dataset, 10% of the training set is used for validation.

### C Baseline Methods

In Appendix C, we introduce baseline methods that are adopted for comparison in the industrial scenario, sequential scenario, non-sequential scenario, and ablation study on searching methods.

[^2]: [http://jmcauley.ucsd.edu/data/amazon/](http://jmcauley.ucsd.edu/data/amazon/)
[^3]: [https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data](https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data)
[^4]: [http://labs.criteo.com/downloads/download-terabyte-click-logs/](http://labs.criteo.com/downloads/download-terabyte-click-logs/)
[^5]: [http://www.kaggle.com/c/avazu-ctr-prediction](http://www.kaggle.com/c/avazu-ctr-prediction)
C.1 Industrial Scenario

The baselines in the industrial scenario include the primitive YouTube DNN [Covington et al., 2016], Wide & Deep [Cheng et al., 2016], DeepFM [Guo et al., 2017] and the more competitive methods of DIN [Zhou et al., 2018b], DIEN [Zhou et al., 2019], DSIN [Feng et al., 2019].

- YouTube DNN [Covington et al., 2016] is usually introduced as the base model in CTR prediction. It utilizes sum pooling on the users’ historical behavior embedding and then concatenates the results with other non-sequential embeddings of the user profile, context profile, and target item/ad profile as the input of MLP to generate the final prediction score.
- Wide & Deep [Cheng et al., 2016] additionally adds a shortcut connection of linear regression on the non-sequential features to the output of MLP to capture both generalization and memorization.
- DeepFM [Guo et al., 2017] replaces the linear regression-based shortcut connection in Wide & Deep with a Factorization Machine to involve feature interactions in the recommendation model.
- DIN [Zhou et al., 2018b] extends the base MLP model by using an attention mechanism on the user behavior with respect to the target item to adaptively learn the diverse characteristic of user interests.
- DIEN [Zhou et al., 2019] improves DIN by using GRU and AUGRU module to the extract user’s evolving interests, where an auxiliary loss is employed to make hidden states more expressive.
- DSIN [Feng et al., 2019] further investigates the homogeneity and heterogeneity in the user behavior and models the user interests through intra-session self-attention layer and inter-session Bi-LSTM layer. The sequential user behaviors are split into sessions by 30-minute intervals, and each session remains at most 10 behaviors.

The aggregation MLP of all baseline methods is fixed to [200, 80], while the activation function is set to Dice in DIN, DIEN, and set to ReLU in other methods. The training is based on the CTR prediction of Eqn. (6) for all methods.

C.2 Sequential Scenario

To evaluate AMEIR in the sequential scenario, we introduce several baselines for comparison. We carefully categorize the state-of-the-art sequential recommender systems and select the representative methods for each category to report the performance in the limited pages of the paper. Since other sequential recommendation methods have been outperformed on the similar datasets by the selected baselines, we omit comparison against them.

- POP is the simplest baseline which only recommends the most popular items with the highest number of interactions from the candidate set (also used for evaluating the complexity of datasets).
- BPR [Rendle et al., 2009] is a legacy method for matrix factorization from implicit feedback and is optimized by a pairwise ranking loss. We use BPR to represent the family of matrix factorization-based methods [He et al., 2017b; Mnih and Salakhutdinov, 2008].
- GRU4Rec+ [Hidasi and Karatzoglou, 2018] is an improved version of GRU4Rec, which utilizes a delicately-designed loss function and sampling strategy to train the GRU model. GRU4REC+ can be seen as the representative of the RNN-based sequential recommender systems, such as GRU4REC [Hidasi et al., 2016], EvoRNN [Belletti et al., 2019], etc.
- STAMP [Liu et al., 2018b] simultaneously captures user’s long-term general interests and short-term current interests with an attention module. STAMP is a typical sequential recommender system without RNN or CNN, representing the MF-based sequential recommender system such as FPMC [Rendle et al., 2010] and TransRec [He et al., 2017a].
- NARM [Li et al., 2017] uses a global GRU encoder to extract sequential behavior and captures the user’s main purpose with the attention mechanism on a local GRU encoder. Regarding the local GRU encoder, the latest 3 items are used for the Beauty dataset and the latest 5 items are used for the Steam dataset. NARM is a representative for the combination of attention and recurrent neural network.
- Caser [Tang and Wang, 2018] is a CNN-based recommender system. It applies horizontal and vertical convolutional operations on the embedding matrix for sequential representation. For Beauty dataset, the number of channels is set to 5 in the vertical convolutions and 2 in the horizontal convolutions. For Steam dataset, the number of channels is set to 8 in the vertical convolutions and 3 in the horizontal convolutions. To ensure a fair comparison, we replace the concatenation of d-dimension embedding in the original design of Caser (as introduced by Eqn. (10) in [Tang and Wang, 2018]) by a summation to avoid the unfair use of the embedding with double dimension.
- NextItNet [Yuan et al., 2019] employs residual blocks of dilated convolutions to increase receptive fields and model both long-range and short-range item dependencies on user behaviors. We set the kernel size of convolutions as 3, and leverage 5 dilated convolutional layers with dilation 1, 2, 1, 2, 2.
- SRGNN [Wu et al., 2019b] applies GNN to generate item representations and then encodes global and current user interests using an attention network. We use the Gated-GNN to learn the item embeddings, add regularization of all parameters with weight 1e-6, and set the initial learning rate to 1e-7 to avoid the gradient explosion of the recurrent unit. SRGNN
is the classical implementation to use GNN on recommendation, which represents the GNN-based recommender system such as GCSAN [Xu et al., 2019], GraphRec [Fan et al., 2019] and PinSage [Ying et al., 2018].

- BERT4Rec [Sun et al., 2019] adopts BERT [Devlin et al., 2019] to learn bi-directional representations of sequences by predicting masked items in the context. The transformer is set to have 2 layers with 4 heads. The dropout rate is set to 0.5 on inputs, 0.2 on feed-forward networks after each multi-head attention and convolution to prevent overfitting. BERT4Rec is a successful transplant of Transformer to recommendation, which is on behalf of Transformer-based recommender systems including SASREC [Kang and McAuley, 2018], MBTREC [Zhang et al., 2020], BST [Chen et al., 2019a], ATRank [Zhou et al., 2018a], etc.

The model-relevant hyper-parameters and initializations are either following the recommended settings in the corresponding papers or tuned by the grid search on the validation sets. We employ the same training process as AMEIR for these methods.

C.3 Non-sequential Scenario

The baseline methods in the non-sequential scenario include representative interaction exploration methods in deep learning-based recommender systems. YouTube DNN [Covington et al., 2016] and Wide & Deep [Cheng et al., 2016] are early attempts of deep learning recommendation methods. DeepFM [Guo et al., 2017], DCN [Wang et al., 2017] and xDeepFM [Lian et al., 2018] enumerate interactions of bounded degrees and combine the feature interactions with MLP via shortcut connection, while IPNN [Qu et al., 2016] and FibiNet [Huang et al., 2019] directly use the feature interactions as the input of MLP. AutoCross [Luo et al., 2019], AutoFIS [Liu et al., 2020] and AutoCTR [Song et al., 2020] are the recently-proposed AutoML-oriented methods. They automatically find useful cross features, FM-based interactions, or aggregation modules in the recommendation model.

- YouTube DNN [Covington et al., 2016] is the base MLP model that takes raw embeddings as the input to learn implicit interactions among the features.
- Wide & Deep [Cheng et al., 2016] extends DNN by introducing shortcut connection of linear regression on the input features to learn both high-order and low-order interactions.
- DeepFM [Guo et al., 2017] replaces the linear regression in Wide & Deep with the FM module.
- IPNN [Qu et al., 2016] combines the inner product of feature interactions with raw feature embeddings as MLP input.
- DCN [Wang et al., 2017] proposes cross network to comprise explicit bit-wise interactions of bounded degrees. The compressed interactions and the output of MLP are concatenated and then fed to a linear layer to generate the prediction scores.
- xDeepFM [Lian et al., 2018] designs compressed interaction network (CIN) to replace cross network in DCN [Wang et al., 2017] by enumerating vector-wise interactions via Hadamard product. The linear regression is also applied for shortcut connection as DeepFM.
- FibiNet [Huang et al., 2019] employs SENet [Hu et al., 2018] to learn the importance of embeddings and then generates bilinear interactions on both SE and original feature embeddings as the input of MLP. The linear regression is also used in FibiNet.
- AutoCross [Luo et al., 2019] learns useful high-order cross feature interactions on a tree-structured space with beam search and successive mini-batch gradient descent. The explored interactions are exploited in the linear part of Wide & Deep.
- AutoFIS [Liu et al., 2020] identifies important feature interactions in the FM model with differentiable architecture search [Liu et al., 2019], and keeps the architecture parameters as attention weights for the corresponding interactions. The generated interactions are applied to the FM module in DeepFM.
- AutoCTR [Song et al., 2020] modularizes representative operations of one-layer MLP, dot product (DP) and factorization machine (FM) to formulate the direct acyclic graphs search space for the recommendation model. It performs an evolutionary algorithm with learning-to-rank guidance during architecture search. The data sub-sampling and hash size reduction are used for acceleration.

To ensure fair comparison, we use the same pre-defined aggregation MLP in AMEIR for the baseline methods, i.e., [200 × 3] for Criteo [Guo et al., 2017] and [500 × 5] for Avazu [Qu et al., 2018] with ReLU activation. Besides, CIN is set to 256 / 128 layers for Criteo / Avazu in xDeepFM. The training is based on the CTR prediction of Eqn. (6) for all methods.

C.4 Ablation Study on Search Methods

The baseline search methods include the brute-force random search, reinforcement learning (RL), evolutionary algorithm (EA), and the popular one-shot methods ENAS, DARTS, SNAS. Conforming to the backbone settings in Sec.3, we select the best performing behavior modeling network, top-5 feature interactions, and top-1 aggregation MLP in different stages for all methods.
• Random [Bergstra and Bengio, 2012] is the baseline NAS method that simply samples architectures from a uniform distribution over the search space. For a fair comparison, we randomly select 300 child models from the search space until the total training time plus the evaluation time reaches 10 times the search cost of the proposed one-shot random search. All child models are trained by 1 epoch according to [Dong and Yang, 2019a; Radosavovic et al., 2020], and the top-5 architectures with the highest validation accuracy will be selected for further evaluation to pick the best one as the final result.

• RL [Zoph and Le, 2017; Zoph et al., 2018] is another baseline method of NAS. At each round, a controller generates a child model from the search space. The child model is then trained and evaluated on the validation set, and the result is treated as the reward to train the weights of the controller for a better outcome. In our implementation, we use a two-layer LSTM as the controller with the hidden size of 32. The controller is trained by REINFORCE rule [Williams, 1992] with Proximal Policy Optimization [Schulman et al., 2017], with a 0.9 exponential moving average of the previous rewards as the baseline [Pham et al., 2018]. We also use the entropy penalty, temperature 5, and the tanh constant 2.5 for the sampling logits [Pham et al., 2018]. Each child model is trained for 1 epoch that is the same as the random search. The searching process is finished once the performance of the consecutively generated models does not grow or reaches the 10-time budgets as the one-shot random search. Similar to the Random, top-5 models in history will be selected for more evaluation.

• EA [Real et al., 2017; Real et al., 2019] maintains a population of child models, and gradually evolves the population for better results. In this paper, we re-implement the regularized evolutionary algorithm [Real et al., 2019] for comparison. Specifically, we first uniformly sample several child models from the search space as the initial population. To mutate the population, we sample one random child model as the parent from the population at each cycle, and then construct a new model from the parent by a mutation, which causes a simple modification on the network architecture, e.g. altering a layer operation in Stage 1, a feature interaction in Stage 2, or an MLP hidden size in Stage 3. Once the child architecture is constructed, it is then trained, evaluated on the validation set, and added to the population. Meanwhile, the oldest member of the population will be removed to maintain the population size. We set the initial population size as 10, and evolve only one architecture at each cycle. Each child model in the population is trained by 1 epoch. Once the population does not change for over 5 cycles or reach the 10-time budgets as the one-shot random search, the algorithm will be finished, and the top-performing models will be derived from the population.

• ENAS [Pham et al., 2018] introduces the one-shot strategy in the RL algorithm. The only difference between ENAS and RL is that the sampled child models in ENAS share the same weights in the one-shot model.

• DARTS [Liu et al., 2019] extends the one-shot strategy by relaxing the discrete search space to be continuous and coupling the architecture parameters with the model parameters to determine the importance of each operation in the one-shot model. The architecture parameters are jointly optimized in the one-shot training along with the model parameters. Specifically, DARTS performs a bi-level optimization, where the model weights and architecture weights are optimized with training data and validation data alternatively. Conforming to the original implementation, we train the architecture parameters with the Adam optimizer and apply early stopping to avoid overfitting on the validation set. The other training settings of the one-shot model is the same as the one-shot random search in AMEIR. After the one-shot modeling training, we retain the strongest operations determined by the architecture parameters as the final result.

• SNAS [Xie et al., 2019] can be regarded as a variant of DARTS. SNAS uses a unified training loss as the optimization target and replaces the softmax function on the architecture parameters in DARTS with the Gumbel-softmax [Jang et al., 2017] to imitate the sampling process on the discrete architecture distribution for better consistency between the one-shot model and the derived model. The other settings in SNAS are the same as DARTS.

• One-shot Random Search [Li and Talwalkar, 2019] is roughly the same as AMEIR, except that the three stages are searched together in a one-shot model and the 6000 randomly selected architectures are used for evaluation to choose the final outcome.

For the fair comparison, we use the same training and evaluation settings in all the search methods.

D Implementation Details

In Appendix D, we introduce the implementation details of AMEIR and baseline methods, including the common settings and the selection of hyper-parameters in industrial, sequential, and non-sequential scenarios.

D.1 Industrial Scenario

For the Alimama dataset in the industrial scenario, we use the embedding size 4 [Feng et al., 2019], batch size 1024, maximum sequence length 50, and Adam optimizer [Kingma and Ba, 2015] for all methods. The embeddings and self-attention layers are initialized uniformly from \([-0.05, 0.05]\); The convolutional block and MLP layers are initialized by He normal [He et al., 2016]; The GRU block is initialized by orthogonal initialization [Saxe et al., 2014]. We apply grid-search to find the best choices of the hyper-parameters in AMEIR and baseline methods. We examine the impact of initial learning rate and regularizer for all
methods from the candidate sets \{1e-3, 1e-4, 1e-5, 1e-6\} and \{1e-3, 1e-4, 1e-5, 1e-6\} respectively. Meanwhile, we attempt to add dropout to both building blocks in the behavior modeling network and hidden layers in the aggregation MLP from \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}.

D.2 Sequential Scenario

For the sequential scenario, we fix the maximum sequence length as 15 for Beauty and 25 for Steam. The initialization is the same as industrial scenario, and the models are optimized by Adam optimization with batch size 512. The input sequences are randomly clipped for the model training, with the last five items served as the positive items and another five randomly sampled items that the user has not interacted with served as the negative items. We also apply grid-search for the hyper-parameter tuning in the sequential scenario. We search the embedding size from \{64, 96, 128, 160, 192\} and find that when the embedding size is greater than 128, the performance of most of the baselines would not further increase and the compared models even become hard to reach convergence. Therefore, we fix the embedding size as 128 for all models. We also search for the initial learning rate from \{1e-3, 1e-4, 1e-5\}, L2 regularizer from \{1e-3, 1e-4, 1e-5, 1e-6\}, and dropout ratio of building blocks from \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}.

D.3 Non-sequential Scenario

For the non-sequential scenario, we use an embedding size of 5 for Criteo and 40 for Avazu. The batch size is set to 4096 and the initialization is the same as the industrial scenario. Similar as the candidate sets in the industrial scenario and sequential scenario, learning rate, L2 regularizer and dropout ratio of MLP layers are chosen from \{1e-3, 1e-4, 3e-5, 1e-5\}, \{1e-3, 1e-4, 1e-5, 1e-6\} and \{0, 0.1, 0.2, 0.3, 0.4, 0.5\} respectively.

E Search Results

In Appendix E, we show the searched building blocks, feature interactions, and aggregation MLPs of AMEIR and the corresponding experimental results in the 4 runs.

E.1 Industrial Scenario

Tab. 9 demonstrates the searched architectures on the industrial dataset. Regarding step 1, AMEIR automatically raises the best behavior modeling network with two or three building blocks based on the sequential data. It can be observed that the searched blocks over the 4 runs all include one non-parameterized operation of average pooling or dummy layer of zero, indicating that the shallow sequential models might be optimal for this dataset. Therefore, compared to the empirical state-of-the-art CTR prediction models such as DIEN and DSIN, the searched building blocks contribute more to the AUC metric. The search results also demonstrate that the one-shot training is capable of finding architectures of appropriate depth to prevent pre-mature convergence of the shallow models and avoid overfitting of the deep models. Besides, although the ”attention from target” operation is involved in the industrial scenario, we find that it does not help improve the AUC result. Regarding step 2, the searched interactions contain both second-order and third-order interactions. Though there are few non-sequential sparse features in the Alimama dataset, the searched interactions can slightly but consistently improve the performance upon the searched blocks. Note that some interactions, e.g., (F04, F10), (F05, F06, F11), and (F06, F11) are shared across different runs. The phenomenon verifies the robustness of AMEIR and enhances the interpretability of the prediction. Regarding step 3, the searched aggregation MLP tends to select the layers with lower dimension activated by ReLU. This decision significantly upgrades the performance on the CTR prediction task, with fewer parameters compared to the models by manual architecture engineering. The prominent performance of the low-dimension MLP enlightens us that a proper MLP could boost the learning capability of a specific architecture, and one-shot searching is able to acquire the best settings of the MLP. Additionally, compared to the novel activations of Swish and Dice, AMEIR prefers ReLU in the searched MLP, which demonstrates that using all ReLUs as the activation functions might be a good starting point for MLP design. The above observations verify the practicability and efficiency of both the three-stage search space and the three-step one-shot searching pipeline of AMEIR.

E.2 Sequential Scenario

Tab. 10 summarizes the searched building blocks in the sequential scenario with the corresponding experimental results. Similar to the industrial scenario, though the maximal number of layers is set to 6, all searched architectures for each dataset contains three to five layers (i.e., at least one “Zero” in each searched architecture), indicating that the shallow models might be more suitable for Beauty and Steam datasets. It is also evident that the one-shot search tends to raise the self-attention block and convolutional block in those two datasets and construct similar architectures to the state-of-the-art manual designs (e.g., #1 of Beauty to NextItNet, #1 of Steam to BERT4REC). Beyond that, the layer normalization is usually deployed on the shallow layers, and the Bi-GRU layer mostly locates at the deep layer. We believe that these placements contribute to the efficient knowledge extraction from user behaviors as well as the fast convergence of sequential modeling.
E.3 Non-sequential Scenario

Tab. 11 depicts the searched interactions, as well as the pre-defined / searched MLP of AMEIR-B and AMEIR-C in the non-sequential scenario. Though AMEIR could search high-order interactions, most of the searched interactions (best-performing interactions) are in second-order while some pairs of features never form appropriate interactions to be selected by the one-shot model according to the statistics. This observation attributes to the automatic recognition of the feasible interactions with appropriate order by the one-shot searching, potentially relieving the effort of manual feature engineering. Same as the industrial scenario, the shared interactions can also be found in the non-sequential scenario, e.g., (C6, C13), (C13, C26) in Criteo and (C15, C18), (device_conn_type, C18) in Avazu, once again verifies the robustness of AMEIR over multiple runs. Regarding the aggregation MLP, we notice that AMEIR-C tends to select MLPs of the higher dimension in the bottom layers and lower dimension in the top layers, which is more reasonable for the neural network design as opposed to the hand-crafted settings in [Guo et al., 2017; Qu et al., 2018]. The adaptive design of MLP further improves the performance on the non-sequential datasets, indicating the effective and efficient one-shot searching of AMEIR. Besides, the preference of ReLU also appears in the non-sequential scenario.

F Discussion on One-shot Strategy

The one-shot strategy has been widely used for accelerating the neural architecture search. In most of the existing literature [Liu et al., 2019; Xie et al., 2019; Wu et al., 2019a; Guo et al., 2019], the one-shot NAS is only applied to derive the best model from the search space, and the derived model is re-trained from scratch. In AMEIR, we find an interesting phenomenon that fine-tuning the derived model with the one-shot model weights can achieve better performance than directly re-training the derived model with random initialization. The phenomenon is aligned with the recently proposed Once-for-All [Cai et al., 2019a], BigNAS [Yu et al., 2020a] and AttentiveNAS [Wang et al., 2020a], which only trains the network once and directly use the one-shot weights for evaluation without re-training. These methods achieve similar or even better performance than training the child models from scratch, which validates our experimental results. Moreover, we observe a larger improvement in using one-shot weights as the initialization in AMEIR. We guess that this is because the recommendation models usually encounter the overfitting problem, and the one-shot training, which is equivalent to a adding large drop-path strategy on the one-shot model, can be regarded as introducing an extra regularization on the model training. Therefore, it helps prevent/delay the appearance of overfitting meanwhile guaranteeing sufficient training of the network. We observe that under similar training settings, one-shot training presents a later overfitting epoch compared to re-training the derived models from scratch, which is consistent with our argument.
Table 9: Searched building blocks, interactions, MLPs and corresponding experimental results of behavior modeling (AMEIR-A), interaction exploration (AMEIR-B) and aggregation MLP investigation (AMEIR-C) on AliMama over 4 runs. Each block $i$ denotes the building block of $i$-th layer, which is represented by either a tuple of (layer operation, activation operation, normalization operation) or a “Zero” operation (skip this layer). Each interaction is represented by a tuple of non-sequential feature fields. Each MLP includes a list of hidden layers, where each hidden layer is represented by a tuple of (hidden size, activation operation).

| Run ID | Searched Blocks | Searched Interactions | Model | MLP | AUC |
|--------|-----------------|-----------------------|-------|-----|-----|
| 1      | Block 1: (1×7 Dconv, Swish, None) Block 2: Zero Block 3: (1×5 Dconv, Identity, LayerNorm) | None | AMEIR-A | [(200, ReLU), (80, ReLU)] | 0.6358 |
|        |                 | ([F08, F11), (F04, F10), (F06, F11), (F07, F09, F10), (F05, F06, F11)] | AMEIR-B |     | 0.6362 |
|        |                 | None | AMEIR-C | [(42, ReLU), (10, ReLU)] | 0.6387 |
| 2      | Block 1: (1×3 Dconv, Identity, None) Block 2: (1×3 AvgPool, Swish, None) Block 3: (1×3 Conv, Identity, None) | None | AMEIR-A | [(200, ReLU), (80, ReLU)] | 0.6357 |
|        |                 | ([F04, F10), (F03, F06), (F06, F07, F11), (F03, F08, F11), (F05, F06, F11)] | AMEIR-B |     | 0.6358 |
|        |                 | None | AMEIR-C | [(42, ReLU), (21, ReLU)] | 0.6379 |
| 3      | Block 1: (2-Head Self Attention, GeLU, None) Block 2: (Bi-GRU, ReLU, LayerNorm) Block 3: (1×3 AvgPool, Identity, None) | None | AMEIR-A | [(200, ReLU), (80, ReLU)] | 0.6356 |
|        |                 | ([F04, F10), (F06, F11), (F01, F06), (F05, F06, F10), (F03, F05, F07)] | AMEIR-B |     | 0.6357 |
|        |                 | None | AMEIR-C | [(42, ReLU), (10, ReLU)] | 0.6383 |
| 4      | Block 1: Zero Block 2: (1×5 Dconv, ReLU, None) Block 3: (Bi-GRU, Identity, LayerNorm) | None | AMEIR-A | [(200, ReLU), (80, ReLU)] | 0.6358 |
|        |                 | ([F08, F11), (F07, F10), (F04, F06), (F06, F10), (F06, F11)] | AMEIR-B |     | 0.6361 |
|        |                 | None | AMEIR-C | [(42, ReLU), (10, ReLU)] | 0.6385 |

* F01: ‘userid’; F02: ‘adgroup_id’; F03: ‘pid’; F04: ‘cms_segid’; F05: ‘cms_group_id’; F06: ‘final_gender_code’; F07: ‘age_level’; F08: ‘pvalue_level’; F09: ‘shopping_level’; F10: ‘occupation’; F11: ‘new_user_class_level’; F12: ‘campaign_id’; F13: ‘customer’; F14: ‘cate_id’; F15: ‘brand’.
Table 10: Searched building blocks and corresponding experimental results of behavior modeling (AMEIR-A) on Beauty and Steam over 4 runs. Each block \( i \) denotes the building block of \( i \)-th layer, which is represented by either a tuple of (layer operation, activation operation, normalization operation) or a “Zero” operation (skip this layer).

| Dataset | Run ID | Searched Blocks                                                                 | HR@1 | HR@5 | NDCG@5 |
|---------|--------|---------------------------------------------------------------------------------|------|------|--------|
| Beauty  | 1      | Block 1: (1 × 5 Dconv, GeLU, LayerNorm)                                        | 0.164| 0.345| 0.260  |
|         |        | Block 2: (1 × 5 Dconv, Identity, None)                                           |      |      |        |
|         |        | Block 3: Zero                                                                    |      |      |        |
|         |        | Block 4: (1 × 7 Dconv, Identity, None)                                           |      |      |        |
|         |        | Block 5: Zero                                                                    |      |      |        |
|         |        | Block 6: (1 × 7 Dconv, Identity, None)                                           |      |      |        |
|         | 2      | Block 1: (4-Head Self Attention, Swish, LayerNorm)                               | 0.153| 0.337| 0.254  |
|         |        | Block 2: (2-Head Self Attention, GeLU, LayerNorm)                               |      |      |        |
|         |        | Block 3: (4-Head Self Attention, Identity, LayerNorm)                            |      |      |        |
|         |        | Block 4: Zero                                                                    |      |      |        |
|         |        | Block 5: (Bi-GRU, Swish, LayerNorm)                                             |      |      |        |
|         |        | Block 6: (1 × 5 Dconv, Swish, None)                                             |      |      |        |
|         | 3      | Block 1: (1 × 3 Conv, ReLU, LayerNorm)                                           | 0.159| 0.341| 0.255  |
|         |        | Block 2: Zero                                                                    |      |      |        |
|         |        | Block 3: Zero                                                                    |      |      |        |
|         |        | Block 4: (4-Head Self Attention, Identity, LayerNorm)                            |      |      |        |
|         |        | Block 5: (4-Head Self Attention, Swish, LayerNorm)                               |      |      |        |
|         |        | Block 6: (1 × 3 Dconv, Swish, None)                                             |      |      |        |
|         | 4      | Block 1: (2-Head Self Attention, Swish, LayerNorm)                               | 0.158| 0.341| 0.256  |
|         |        | Block 2: (4-Head Self Attention, Swish, LayerNorm)                               |      |      |        |
|         |        | Block 3: (1 × 3 Dconv, ReLU, None)                                               |      |      |        |
|         |        | Block 4: (Bi-GRU, Swish, LayerNorm)                                             |      |      |        |
|         |        | Block 5: Zero                                                                    |      |      |        |
|         |        | Block 6: (Bi-GRU, Swish, None)                                                   |      |      |        |
| Steam   | 1      | Block 1: Zero                                                                    | 0.339| 0.682| 0.517  |
|         |        | Block 2: (4-Head Self Attention, Swish, LayerNorm)                               |      |      |        |
|         |        | Block 3: (4-Head Self Attention, Identity, LayerNorm)                            |      |      |        |
|         |        | Block 4: Zero                                                                    |      |      |        |
|         |        | Block 5: (1 × 5 Dconv, GeLU, None)                                              |      |      |        |
|         |        | Block 6: (1 × 5 Dconv, GeLU, None)                                              |      |      |        |
|         | 2      | Block 1: (2-Head Self Attention, GeLU, LayerNorm)                               | 0.339| 0.683| 0.519  |
|         |        | Block 2: Zero                                                                    |      |      |        |
|         |        | Block 3: (4-Head Self Attention, Identity, LayerNorm)                            |      |      |        |
|         |        | Block 4: Zero                                                                    |      |      |        |
|         |        | Block 5: Zero                                                                    |      |      |        |
|         |        | Block 6: (Bi-GRU, Identity, None)                                               |      |      |        |
|         | 3      | Block 1: (1 × 1 Conv, GeLU, None)                                               | 0.337| 0.681| 0.517  |
|         |        | Block 2: (1 × 3 Conv, ReLU, LayerNorm)                                           |      |      |        |
|         |        | Block 3: (4-Head Self Attention, ReLU, LayerNorm)                               |      |      |        |
|         |        | Block 4: (2-Head Self Attention, GeLU, None)                                     |      |      |        |
|         |        | Block 5: Zero                                                                    |      |      |        |
|         |        | Block 6: (2-Head Self Attention, Swish, None)                                    |      |      |        |
|         | 4      | Block 1: Zero                                                                    | 0.335| 0.678| 0.515  |
|         |        | Block 2: (1 × 7 Dconv, GeLU, LayerNorm)                                          |      |      |        |
|         |        | Block 3: Zero                                                                    |      |      |        |
|         |        | Block 4: (4-Head Self Attention, Swish, None)                                    |      |      |        |
|         |        | Block 5: (2-Head Self Attention, Swish, None)                                    |      |      |        |
|         |        | Block 6: (1 × 5 Dconv, ReLU, None)                                              |      |      |        |
Table 11: Searched interactions, MLPs and corresponding experimental results of interaction exploration (AMEIR-B) and additional aggregation MLP investigation (AMEIR-C) on Criteo and Avazu over 4 runs. Each interaction is represented by a tuple of non-sequential feature fields. Each MLP includes a list of hidden layers, where each hidden layer is represented by a tuple of (hidden size, activation operation).

| Dataset | Run ID | Search Interactions                                                                 | Model       | MLP                                                                 | AUC  | Log Loss |
|---------|--------|-------------------------------------------------------------------------------------|-------------|----------------------------------------------------------------------|------|----------|
| Criteo  | 1      | [(C10, C13), (C13, C26), (C15, C24), (C15, C20), (C11, C21), (C13, C19), (C4, C16), (C2, C15), (C7, C19), (C21, C26), (C19, C24), (C15, C19), (C14, C23), (C3, C6), (C10, C16)] | AMEIR-B     | [(200, ReLU), (200, ReLU), (200, ReLU)]                            | 0.8029 | 0.4487   |
|         |        |                                                                                     | AMEIR-C     | [(174, Swish), (21, ReLU), (21, Identity)]                          | 0.8034 | 0.4482   |
|         | 2      | [(C6, C13), (C11, C24), (C4, C26), (C13, C26), (C2, C24), (C11, C20), (C6, C16), (C7, C16), (C10, C19), (C2, C15), (C3, C18), (C10, C24), (C4, C25), (C3, C21), (C16, C17)] | AMEIR-B     | [(200, ReLU), (200, ReLU), (200, ReLU)]                            | 0.8030 | 0.4486   |
|         |        |                                                                                     | AMEIR-C     | [(152, ReLU), (87, Identity), (21, ReLU)]                           | 0.8035 | 0.4483   |
|         | 3      | [(C6, C13), (C10, C13), (C7, C11), (C11, C24), (C4, C26), (C13, C26), (C2, C24), (C11, C23), (C3, C19), (C3, C12), (C2, C15), (C10, C24), (C19, C24), (C10, C13), (C6, C11), (C2, C13), (C15, C20)] | AMEIR-B     | [(200, ReLU), (200, ReLU), (200, ReLU)]                            | 0.8032 | 0.4483   |
|         |        |                                                                                     | AMEIR-C     | [(152, ReLU), (65, ReLU), (21, ReLU)]                              | 0.8036 | 0.4480   |
|         | 4      | [(C6, C13), (C13, C26), (C14, C18), (C12, C16), (C15, C20), (C11, C23), (C3, C19), (C12, C13), (C6, C11), (C2, C13), (C15, C20)] | AMEIR-B     | [(200, ReLU), (200, ReLU), (200, ReLU)]                            | 0.8028 | 0.4490   |
|         |        |                                                                                     | AMEIR-C     | [(174, Swish), (65, Dice), (21, Dice)]                             | 0.8032 | 0.4485   |
| Avazu   | 1      | [(device_conn_type, C18), (app_category, device_type), (site_category, C18), (app_domain, C15), (site_id, C21), (app_category, C15), (site_id, dev_id, id, app_id, C21), (C1, C15), (weekday, site_id), (app_domain, C17), (device_id, C15), (day, app_category), (day, device_model)] | AMEIR-B     | [(500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU)] | 0.7794 | 0.3796   |
|         |        |                                                                                     | AMEIR-C     | [(1092, Swish), (780, ReLU), (156, Identity), (156, ReLU), (156, Identity)] | 0.7798 | 0.3794   |
|         | 2      | [(device_conn_type, C18), (C1, C21), (site_domain, dev_id, C21), (app_category, C15), (site_id, C20), (device_model, C16), (site_category, C21), (site_category, device_domain, id), (weekday, site_id), (app_id, C17), (device_id, C15), (site_id, device_id), (day, app_category)] | AMEIR-B     | [(500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU)] | 0.7796 | 0.3795   |
|         |        |                                                                                     | AMEIR-C     | [936, Identity], (468, Dice), (156, ReLU), (156, Swish), (156, ReLU)] | 0.7799 | 0.3793   |
|         | 3      | [(C15, C18), (day, C16), (banner_pos, app_id), (C1, C21), (app_id, device_type), (device_id, C16), (banner_pos, device_type), (site_category, device_type), (banner_pos, C20), (app_id, device_id), (site_id, C15), (weekday, site_id), (device_id, C15)] | AMEIR-B     | [(500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU)] | 0.7802 | 0.3793   |
|         |        |                                                                                     | AMEIR-C     | [936, ReLU], (468, Identity), (156, Identity), (156, ReLU), (156, ReLU)] | 0.7806 | 0.3791   |
|         | 4      | [(device_conn_type, C18), (app_category, device_type), (day, C1), (device_id, C16), (weekday, site_domain), (banner_pos, C20), (app_domain, C15), (day, weekday), (site_id, C20), (app_category, device_conn_type), (hour, C14), (app_domain, device_ip), (site_domain, device_ip), (weekday, site_id), (C16, C21)] | AMEIR-B     | [(500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU), (500, ReLU)] | 0.7797 | 0.3792   |
|         |        |                                                                                     | AMEIR-C     | [936, ReLU], (312, ReLU), (156, ReLU), (156, Identity)] | 0.7802 | 0.3790   |