# AutoGroup: Automatic Feature Grouping for Modelling Explicit High-Order Feature Interactions in CTR Prediction*

Bin Liu[†*], Niannan Xue[‡*], Huifeng Guo[†], Ruiming Tang[†],
Stefanos Zafeiriou[‡], Xiuqiang He[†], Zhenguo Li[†]
[†]Huawei Noah's Ark Lab [‡]Imperial College London
liubinbh@126.com, sparrowxue@hotmail.com, s.zafeiriou@imperial.ac.uk
{huifeng.guo, tangruiming, hexiuqiang1, li.zhenguo}@huawei.com

## ABSTRACT

Modelling feature interactions is key in Click-Through Rate (CTR) predictions. State-of-the-art models usually include explicit feature interactions to better model non-linearity in a deep network, but enumerating all feature combinations of high orders is not efficient and brings challenges to network optimization. In this work, we use AutoML to seek useful high-order feature interactions to train on without manual feature selection. For this purpose, an end-to-end model, AutoGroup, is proposed, which casts the selection of feature interactions as a structural optimization problem. In a nutshell, AutoGroup first automatically groups useful features into a number of feature sets. Then, it generates interactions of any order from these feature sets using a novel interaction function. The main contribution of AutoGroup is that it performs both dimensionality reduction and feature selection which are not seen in previous models. Offline experiments on three public large-scale benchmark datasets demonstrate the superior performance and efficiency of AutoGroup over state-of-the-art models. Furthermore, a ten-day online A/B test verifies that AutoGroup can be reliably deployed in production and outperform the commercial baseline by 10% on average in terms of CTR and CVR.

## CCS CONCEPTS

• **Information systems → Recommender systems**.

## KEYWORDS

Neural Networks; Recommender Systems; Deep Learning; High-Order Feature Interactions; AutoML

**ACM Reference Format:**
Bin Liu[†*], Niannan Xue[‡*], Huifeng Guo[†], Ruiming Tang[†], and Stefanos Zafeiriou[‡], Xiuqiang He[†], Zhenguo Li[†]. 2020. AutoGroup: Automatic Feature Grouping for Modelling Explicit High-Order Feature Interactions in CTR Prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401082

*Co-first authors with equal contributions. Bin Liu is now affiliated with ByteDance. Ruiming Tang is the corresponding author.


*SIGIR '20, July 25–30, 2020, Virtual Event, China*

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401082

## 1 INTRODUCTION

Click-Through Rate (CTR) prediction is a crucial task in recommender systems, which estimates the probability of a user clicking on a given item [9, 27]. The revenue of an online advertising system depends critically on the performance of CTR prediction [9, 26] because candidate advertisements are ranked by CTR × bid, where bid is the price to pay once the advertisement is clicked [9, 39, 40].

The key challenge in CTR prediction is how to effectively model feature interactions [9, 18]. Models based on quadratic polynomial mappings [3] and factorization machines [14, 29] learn $2^{\text{nd}}$-order feature interactions while Higher-Order Factorization Machine (HOFM) [2] approximates high-order feature interactions through the ANOVA kernel. Recently, deep learning models showed significant improvements over traditional models in recommender systems [9, 16, 18, 27, 40]. Theoretically, MLP is able to model any arbitrary order of feature interactions [12]. However, as shown in [1], it requires many more parameters than tensor factorization models to approximate high-order interactions, which makes training more difficult. As an enhancement to MLP, including low-order feature interactions explicitly in a deep learning model achieves better performance [1]. However, it is both time and space consuming to explicitly enumerate $p^{\text{th}}$-order ($p \geq 3$) feature interactions as the number of interaction terms is $\binom{m}{p}$, where $m$ is the number of fields in the input data. Therefore, most methods are limited to $2^{\text{nd}}$-order feature interactions only, such as DeepFM [9], PNN [26] and PIN [27]. Existing models that model explicit high-order feature interactions, such as CrossNet [32] and xDeepFM [16], simply enumerate all high-order interactions. This approach has two drawbacks: first, including unnecessary interactions may have a detrimental effect on the model's performance; second, it makes it difficult for neural networks to learn an optimal interaction set in such a huge search space. Indeed, it has been found recently [18] that explicitly modelling high-order feature interactions does not work as expected. It can even perform worse than PIN [27], which only explicitly models $2^{\text{nd}}$-order feature interactions.

In real-world scenarios, useful feature interactions are usually sparse relative to the combination space of raw features [27]. Therefore, to model high-order feature interactions effectively, one has to identify and select useful feature interactions. Inspired by the recent progress on AutoML, we model the selection of feature interactions as a structural optimization problem. Specifically, we

propose a novel method, **AutoGroup**, to **Auto**matically **Group** features together, whose interactions are useful to CTR prediction. It is because of this sparse selection of useful feature interactions that the learning difficulty of neural networks is greatly alleviated.

Modelling with AutoGroup consists of three stages: the *Automatic Feature Grouping Stage*, the *Interaction Stage*, and the *MLP Stage*. In the *Automatic Feature Grouping Stage*, the selection of useful feature interactions at a given order is treated as a structural optimization problem. We use AutoML techniques to generate useful feature interactions from a relatively small but effective feature set in a data-driven manner. Meanwhile, an ensemble [17] of feature sets are used by AutoGroup for each order of feature interactions to improve generalizability. In the *Interaction Stage*, a novel interaction function is used to construct the $p^{\text{th}}$-order interaction from the feature sets. Feature interactions of interested orders are generated separately. Lastly, in the *MLP Stage*, the generated interactions at different orders are concatenated together before feeding into a multilayer perceptron (MLP) for further learning.

Experimental results on three large-scale public benchmark datasets demonstrate the superior performance of AutoGroup over the state-of-the-art baselines. Ablation study shows that prediction performance is improved as high-order feature interactions are added into the model. Moreover, comparison between AutoGroup and the feature selection method using regularization confirms that AutoGroup is more accurate and more stable. Efficiency analysis reveals that AutoGroup requires less training and inference time than the best two state-of-the-art models (PIN [27] and FGCNN [18]) , which makes it more suitable for deployment in commercial recommender systems. Last but not the least, we deploy AutoGroup in a mainstream online app recommender system. The main contributions are highlighted as follows:

- A novel perspective in modelling explicit high-order feature interactions is studied: instead of simply enumerating all feature interactions as existing works do, we propose to identify and select useful feature interactions.
- We propose an end-to-end model, AutoGroup, which treats the selection process of high-order feature interactions as a structural optimization problem, and solves it with AutoML techniques. High-order feature interactions are modelled by a novel interaction function on the selected feature sets.
- Extensive offline experiments on three large-scale public benchmarks show the superior performance and efficiency of AutoGroup over the state-of-the-art algorithms. We deploy AutoGroup in a mainstream app recommender system. On average, AutoGroup achieves 10% improvement in terms of CTR and CVR over a strong baseline in a ten-day A/B test.

The rest of this paper is organized as follows. We first discuss related work in Section 2. Then, in Section 3, the AutoGroup model is introduced and explained in detail. Offline and online experimental results are presented and discussed in Section 4. Finally, Section 5 concludes the paper.

## 2 RELATED WORK

### 2.1 Traditional Models for CTR Prediction

Logistic Regression (LR) models such as FTRL [23] are widely used in CTR prediction for their simplicity and efficiency. Human efforts are usually needed for LR models to learn feature interactions [9, 31]. Poly2 [3] enumerates all pairwise feature interactions to avoid feature engineering which works well on dense data. For sparse data, Factorization Machine (FM) and its variants [24, 29] project each feature into a low-dimensional vector and models feature interactions by inner product while FFM [14] enables each feature to have multiple latent vectors to interact with features from different fields. However, both FM and FFM can only model $2^{\text{nd}}$-order feature interactions. An efficient algorithm for training arbitrary-order Higher-Order FM (HOFM) was proposed in [2] by introducing the ANOVA kernel. As shown in [34], HOFM achieves only marginal improvement over FM whereas using many more parameters.

### 2.2 Deep Learning Models for CTR Prediction

Recently, deep learning models have achieved state-of-the-art performance on some public benchmarks for recommendation tasks [18, 27]. Several models use an MLP to improve FM, including Attention FM [34] and Neural FM [10]. FNN [37] uses FM to pre-train feature embeddings and then feeds them into an MLP. Wide & Deep Learning [5] jointly trains a wide model for artificial features and a deep model for raw features. To avoid feature engineering, DeepFM [9] uses a FM layer to replace the wide component in Wide & Deep. IPNN [27] (also known as PNN [26]) feeds the interaction result of the FM layer and feature embeddings into an MLP, leading to good performance. PIN [27] introduces a micro-network for each pair of fields to model pairwise feature interactions rather than using simple inner product as in PNN and DeepFM. Due to the curse of dimensionality, PNN, DeepFM and PIN cannot explicitly model higher-order feature interactions, which could further improve model performance. CrossNet [32] introduces a cross operation to learn explicit feature interactions. However, as stated in [16], such a cross operation can not learn effective interactions since the output of cross networks is a scalar multiplication of raw embeddings. xDeepFM [16] uses a Compressed Interaction Network (CIN) to enumerate and compress all feature interactions, for modelling an explicit order of interactions. However, it uses so many parameters that great challenges are posed to identify important feature interactions in the huge combination space. FGCNN [18] combines a CNN and MLP to generate new features for feature augmentation. It achieves state-of-the-art performance on various datasets, but is too complex to be applied in industrial applications.

In this paper, instead of enumerating all feature interactions as existing works do, AutoGroup utilizes AutoML techniques to automatically identify and select important feature interactions from a large combination space.

### 2.3 Neural Architecture Search

The success of deep learning is accompanied by a rising demand for architecture engineering [7, 21]. Neural Architecture Search (NAS) is a logical next step in automating machine learning. There are basically three existing frameworks for neural architecture search [8, 19]: Evolution-based NAS [28], Reinforcement-learning based NAS [25], and gradient-based NAS [20, 33, 35, 38]. We focus on the last one for its efficiency. DARTS [20] proposed a gradient-based method where attention on operations is used to analytically calculate the weight of each operation. After the convergence of

the parent network, DARTS removes operations with small attention scores. However, it introduces untractable bias to the loss function and causes inconsistency between the performance of the derived child network and converged parent network [35]. Instead of optimizing real-valued weights on the operations as in DARTS, SNAS [35] optimizes a distribution over the candidate operations, which employs the distribution reparametrization [13] to relax the discrete distribution and makes it differentiable, enabling optimization via gradient descent [7]. Inspired by SNAS, we model the selection process of important feature interactions as a structural learning problem. AutoGroup integrates AutoML into recommender systems, enabling efficient and effective high-order feature interactions.

# 3 AUTOMATIC FEATURE GROUPING

## 3.1 Overview

In this section, we describe the proposed Automatic Feature Grouping (AutoGroup) model in detail. As presented in Figure 1, AutoGroup consists of three stages: *Automatic Feature Grouping Stage*, *Interaction Stage*, and *MLP Stage*. As discussed in Section 1, the essential challenge in modelling high-order feature interactions is to identify important feature interactions, instead of enumerating them all. To tackle such a challenge, in the *Automatic Feature Grouping Stage*, we group features into a number of feature sets such that the feature interactions generated from individual groups are effective at a given order. The feature grouping process is cast as a structural optimization problem. For a specific order, multiple feature sets are selected to generate feature interactions for better generalizability. Then, in the *Interaction Stage*, a feature set is represented as the weighted sum of the feature embeddings in this set. We propose a novel interaction function on the representation of a feature set to generate feature interactions at a given order. In the *MLP Stage*, feature interactions at individual orders are concatenated and then fed to an MLP to learn further feature interactions. The details of these stages are elaborated in the following subsections. Before introducing them, we present some preliminaries.

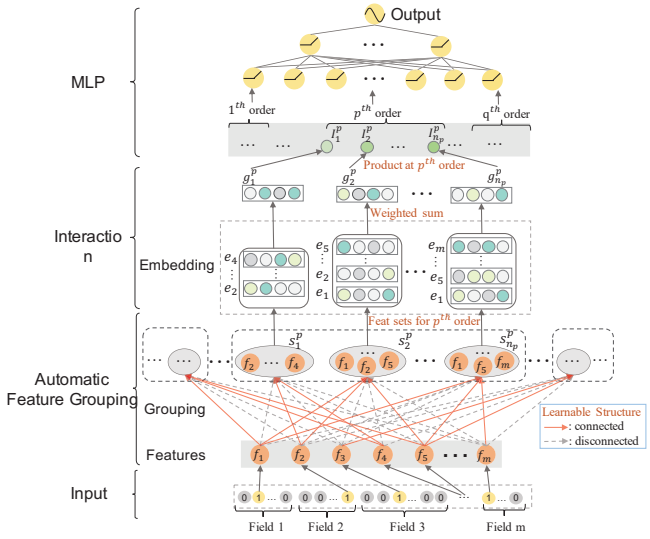


**Figure 1: Overview of AutoGroup**

## 3.2 Feature Embeddings in CTR Prediction

In most CTR prediction tasks, data is collected in a multi-field categorical form[1] [27, 32, 36]. Each data instance is transformed into a high-dimensional sparse (binary) vector via one-hot-encoding [11]. For example, the raw data instance (Gender=Male, Height=185, Age=18, Name=Bob) can be represented as:

$$\underbrace{(0,1)}_{\text{Gender=Male}}\ \underbrace{(0,\ldots,1,0,0)}_{\text{Height=185}}\ \underbrace{(0,1,\ldots,0,0)}_{\text{Age=18}}\ \underbrace{(1,0,0,..,0)}_{\text{Name=Bob.}}$$

An embedding layer is applied to compress the raw features to low-dimensional vectors before feeding them into neural networks. For a univalent field, (e.g., "Gender=Male"), its embedding is the feature embedding; for a multivalent field (e.g., "Interest=Football, Basketball"), the field embedding is the average of the feature embeddings [6] [2]. More formally, in an instance, each field $f_i$ ($1 \le i \le m$) is represented as a low-dimensional vector $e_i \in R^{1\times k}$, where $m$ is the number of fields and $k$ is the embedding size. Therefore, each instance can be represented as an embedding matrix $E = (e_1^\top, e_2^\top, ..., e_m^\top)^\top \in R^{m\times k}$.

## 3.3 Automatic Feature Grouping Stage

*3.3.1 Intuition.* The key challenge in CTR prediction is modelling feature interactions. Traditional methods (e.g., FM [29], FFM [14]) and deep learning models (e.g., DeepFM [9], PIN [27]) enumerate 2$^{\text{nd}}$-order feature interactions. In real-world scenarios, feature interactions at high orders may have a significant impact on prediction [16]. For example, a feature combination of (Gender=Male, age<30, hour_time>18) is an important 3$^{\text{rd}}$-order feature interaction to predict whether a user will download a game App. However, the number of the $p^{\text{th}}$-order feature interactions is $\binom{m}{p}$, which is both time and space consuming to enumerate when $p \ge 3$ since $m$ often ranges from 20 to 50. Some deep learning models (e.g., xDeepFM [16] and CrossNet [32]) perform enumeration to include high order feature interactions in neural networks. Such a brute-force strategy makes neural networks hard to optimize and leads to even inferior performance compared to the models with just explicit 2$^{\text{nd}}$-order interactions (e.g., PIN, PNN) [18].

Useful feature interactions are usually sparse in the large combination space. Therefore, the simple strategy of enumerating all feature interactions makes neural networks difficult to achieve the optimal set of network weights, and hard to identify such useful interactions. To tackle such a challenge, we propose to identify important feature interactions through a data-driven manner, so that neural networks no longer need to include useless interactions during training and can achieve better performance. The main idea is to find multiple small sets of features, where the feature interactions in each such identified feature set are effective at a given order. The representations of all selected feature sets are combined to become the representation of $p^{\text{th}}$-order feature interactions.

More formally, given input features $f_i$ ($1 \le i \le m$) where $f_i$ is the feature value for the $i^{\text{th}}$ field and $m$ is the number of features, to represent $p^{\text{th}}$-order feature interactions, we select $n_p$ feature sets

[1] Numerical features are usually transformed into categorical form by bucketing.
[2] In Figure 1, if field $i$ is multivalent, then $f_i$ represents the multivalent features and $e_i$ is the averaged embedding of features in $f_i$ [6].

**Table 1: Main Notations**

| Notation | Meaning |
|---|---|
| $n$ | the number of features |
| $m$ | the number of fields |
| $k$ | embedding size of features |
| $q$ | the maximum order of feature interactions |
| $f_i$ | the feature value for field $i$ |
| $e_i$ | embedding for field $i$ |
| $n_p$ | the number of feature sets in the $p^{\text{th}}$-order |
| $s_j^p$ | the $j^{\text{th}}$ feature set for the $p^{\text{th}}$-order |
| $\Pi_{i,j}^p$ | an indicator whether $f_i$ is selected into set $s_j^p$ |
| $\bar{\Pi}_{i,j}^p$ | a relaxation of $\Pi_{i,j}^p$ to transform it into a continuous space |
| $\alpha_{i,j}^p$ | the logit to calculate $\bar{\Pi}_{i,j}^p$ |
| $\Pi^p$ | the set $\{\Pi_{i,j}^p \mid 1 \le i \le m, 1 \le j \le n_p\}$ |
| $g_j^p$ | the representation for feature set $s_j^p$ |
| $I_j^p$ | the interaction of feature set $s_j^p$ at $p^{\text{th}}$-order |
| $H_i$ | the number of hidden neurons in the $i^{\text{th}}$ layer of MLP |
| $h$ | the number of hidden layers in MLP |

$s_j^p$ $(1 \le j \le n_p)$, where $n_p$ is a hyper-parameter, such that $p^{\text{th}}$-order feature interactions can improve prediction performance.

*3.3.2 Method for Automatic Feature Grouping.* Specifically, a variable $\Pi_{i,j}^p$ is introduced to indicate whether feature $f_i$ is selected into feature set $s_j^p$ at the $p^{\text{th}}$-order. The goal is to find the best selection of feature sets that can minimize the training loss:

$$\min_{\Pi,\theta} \mathcal{L}_{train}(Y, \mathcal{F}(\Pi^1, ..., \Pi^p, ..., \Pi^q, X, \theta)) \tag{1}$$

where $q$ is the maximum order of feature interactions, $\Pi^p$ is the selection of feature sets at the $p^{\text{th}}$-order, $\theta$ denotes the remaining parameters in the prediction model $\mathcal{F}$, $X$ is the one-hot representation of training samples and $Y$ is the set of corresponding labels. As shown in Figure 1, the selection of feature sets can be viewed as a structural optimization problem: each feature is allowed to be selected by any feature set where each feature set groups related features together such that their interaction is likely to improve prediction performance. Inspired by DARTS (Differentiable Architecture Search [20]), which makes the discrete search space continuous, we relax the binary choice of a feature being selected by a feature set to a softmax over the two possibilities, as follows:

$$\bar{\Pi}_{i,j}^p = \frac{1}{1+\exp(-\alpha_{i,j}^p)}\Pi_{i,j}^p + \frac{\exp(-\alpha_{i,j}^p)}{1+\exp(-\alpha_{i,j}^p)}(1-\Pi_{i,j}^p) \tag{2}$$

where $\Pi_{i,j}^p \in \{0,1\}$ is the binary choice of whether feature $f_i$ is selected into feature set $s_j^p$. The relaxation value $\bar{\Pi}_{i,j}^p$ takes value 1 (i.e., feature $f_i$ is selected) with probability $\frac{1}{1+\exp(-\alpha_{i,j}^p)}$ and 0 (i.e., $f_i$ is not selected) with probability $\frac{\exp(-\alpha_{i,j}^p)}{1+\exp(-\alpha_{i,j}^p)}$. After the relaxation, the task of feature grouping reduces to learning a set of continuous variables $\alpha = \{\alpha_{i,j}^p\}$. We call such variables $\alpha$ *structural parameters*, as they are utilized to determine network architecture.

In AutoGroup, we learn the structural parameters $\alpha = \{\alpha_{i,j}^p\}$ directly from data. However, it is difficult to get the derivative of the probabilistic selection action, which samples the binary choice from a given probabilistic distribution for each feature $f_i$ in each feature set $s_j^p$. With the great progress in the area of Neural Architecture Search [20], the derivative can be approximated by introducing the Gumbel-Softmax trick [13]. It replaces the non-differentiable sample from a categorical distribution with a differentiable sample from a novel Gumbel-Softmax distribution which can be smoothly annealed into a categorical distribution.

**Algorithm 1:** Implementation for differentiable selections

**Input:** softmax values: $(\bar{\Pi}_{i,j}^p)_0, (\bar{\Pi}_{i,j}^p)_1$
**Output:** differentiable operation choice over input: o

```
set p = [(Π̄^p_{i,j})_0, (Π̄^p_{i,j})_1];
set c = [0, 0];
if p_0 >= p_1 then
    c[0] = 1
else
    c[1] = 1
end
o = stop_gradient[3] ( c - p ) + p
```

More specifically, we can reparameterize $\bar{\Pi}_{i,j}^p$ as:

$$\bar{\Pi}_{i,j}^p = argmax_{o\in\{0,1\}}(\log \alpha_o + G) \tag{3}$$

where $\alpha_1 = \frac{1}{1+\exp(-\alpha_{i,j}^p)}$, $\alpha_0 = 1 - \alpha_1$ and $G$ is a random variable which follows the standard Gumbel distribution such that $G = -\log(-\log u)$ with $u \sim U(0,1)$ where $U$ is the uniform distribution. With the random variable sampled from the standard Gumbel distribution, we can get variables from the desired distribution in Eq. 3 which is proved in [22]. However, it is still non-differentiable due to the *argmax* operation. To make the operation differentiable, we relax Eq. 3 with a softmax:

$$(\bar{\Pi}_{i,j}^p)_o = \frac{\exp((\log \alpha_o + G_o)/\tau)}{\sum_{i\in\{0,1\}}\exp((\log \alpha_i + G_i)/\tau)} \tag{4}$$

where $\tau > 0$ is a temperature parameter such that when $\tau \to 0$, $\bar{\Pi}_{i,j}^p$ is one-hot and the Gumbel-softmax distribution is identical to the distribution in Eq. 3. With Eq. 4, we are able to make the binary decision by taking the selection operation with the larger softmax value, which can be implemented in a differentiable way as Algorithm 1. To summarize, with the Gumbel-Softmax trick, the process of grouping features into different sets is trainable via back-propogation.

We generate feature sets at different orders, according to the probability function in Eq. 4. At the beginning, the probabilities of the binary choices are initialized to be equal, i.e., $(\bar{\Pi}_{i,j}^p)_0 = (\bar{\Pi}_{i,j}^p)_1$. During the training process, if the $p^{\text{th}}$-order interactions in each feature set are helpful for the prediction model, the selection probabilities will be updated. Although $\bar{\Pi}_{i,j}^p$'s are independent probabilities, their probabilities will be jointly optimized to learn an optimal grouping strategy to reduce the training loss with the help of $p^{\text{th}}$-order interactions.

**Note:** During testing, the Gumbel-softmax trick is no longer used, which means that we make the selection action based on the maximum probability between $a_1 = \frac{1}{1+\exp(-\alpha_{i,j}^p)}$ and $a_0 =$

[3] In tensorflow, we can use tf.stop_gradients to redirect the gradients from $o$ to $p$.

$\frac{\exp(-\alpha^p_{i,j})}{1+\exp(-\alpha^p_{i,j})}$ (if $a_1 > a_0$: select $f_i$ into the $j^{\text{th}}$ feature set of the $p^{\text{th}}$-order; otherwise, do not select).

## 3.4 Interaction Stage

*3.4.1 Feature Set Representation.* After feature grouping in the previous stage, $n_p$ feature sets are generated for the $p^{\text{th}}$-order. To make the computations in the *Interaction Stage* efficient, the representation of a feature set is defined as a linear function of the included features. Specifically, the representation of a feature set is the weighted sum of representations of the features in this set. Mathematically, the representation of a feature set $s^p_j$ is defined as:

$$g^p_j = \sum_{f_i \in s^p_j} w^p_i e_i \tag{5}$$

where $e_i$ is the embedding of feature $f_i$ and the weights $w^p_i$ are learned as network weights. Finding optimal $p^{\text{th}}$-order interactions in the $p^{\text{th}}$-order combination space is NP-hard, and therefore we do not fix the number of features in each feature set at any given order.

*3.4.2 Feature Interaction at a given order.* In the previous *Automatic Feature Grouping Stage*, for the $p^{\text{th}}$-order, $n_p$ feature sets $s^p_j$ ($j \in [1, n_p]$) are generated, so that the $p^{\text{th}}$-order interaction of the selected features in each feature set are useful to prediction performance. Afterwards, the $p^{\text{th}}$-order interaction is constructed using the representation $g^p_j$ of each feature set $s^p_j$.

The design of $p^{\text{th}}$-order feature interaction is inspired by the reformulation of Factorization Machine (FM) [30]:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < e_i, e_j > x_i x_j \tag{6}$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\left( (\sum_{i=1}^{n} x_i e_i)^2 - \sum_{i=1}^{n} (x_i e_i)^2 \right) \tag{7}$$

where $x_i$ is the one-hot vector of feature $f_i$; $e_i$ is the embedding vector of feature; $< \cdot, \cdot >$ is the inner product of two vectors and the $(\cdot)^2$ is the inner product of a vector with itself. Note that the reformulation (Eq. 7) can reduce the time complexity of the standard formulation (Eq. 6) from $o(n^2)$ to $o(n)$. So, in this paper, we generalize the reformulation (Eq. 7) for the $p^{\text{th}}$-order interaction as

$$\left( (\sum_{i=1}^{n} x_i e_i)^p - \sum_{i=1}^{n} (x_i e_i)^p \right), \tag{8}$$

which is different from Eq. 8 in [30] for the higher-order Factorization Machine but more desirable because it can reduce the complexity from the impractical $o(n^p)$ to $o(n)$ while keeping all the $p^{\text{th}}$-order interaction terms.

Nevertheless, Eq. 8 contains "unexpected" terms. For instance, $(e_i)^2 e_j$ is considered when modelling the $3^{\text{rd}}$-order interactions. Such terms are unavoidable when the efficient estimation of the $p^{\text{th}}$-order interaction is calculated using Eq. 8. However, these "unexpected" terms can be viewed as the $p'^{\text{th}}$-order interactions (where $p' < p$), which can also be beneficial to the model. In our framework, we adapt Eq. 8 to the following formula to calculate the $p^{\text{th}}$-order interaction for each feature set $s^p_j$:

$$I^p_j = \begin{cases} (g^p_j)^p - \sum_{f_i \in s^p_j} (w^p_i e_i)^p \in R, & p \geq 2 \\ g^p_j \in R^k, & p = 1 \end{cases} \tag{9}$$

where $(g^p_j)^p = \sum_{t=1}^{k} \overbrace{g^p_{j,t} \cdot \ldots \cdot g^p_{j,t}}^{p}$ and $(g^p_j)^p$ is the element-wise product of $g^p_j$ with itself p times after which a sum over its k elements is taken. Because $g^p_j$ is the weighted sum of the selected features (Eq. 6), $(g^p_j)^p$ contains all $p^{\text{th}}$-order interactions of the selected features. For example, if features $e_{\text{device\_ip}}$, $e_{\text{app\_id}}$, $e_{\text{site\_domain}}$, $e_{\text{site\_id}}$ have been selected into a feature set, then the $3^{\text{rd}}$-order interaction $(e_{\text{device\_ip}} + e_{\text{app\_id}} + e_{\text{site\_domain}} + e_{\text{site\_id}})^3$ contains all $3^{\text{rd}}$-order interactions between the four features.

$I^p_j$ sums $p^{\text{th}}$-order interactions in $s^p_j$ while excluding the normalization terms. If a feature set $s^p_j$ contains less features than the order that it is supposed to model (namely, $|s^p_j| < p$), the maximum order that $s^p_j$ can model is degenerated to $|s^p_j|$, instead of $p$.

## 3.5 MLP Stage

After the interaction stage, the interaction results of the feature sets at each order are generated. To learn further implicit feature interactions, we concatenate the interaction results of each order and feed them into an MLP. We denote the input of the $i^{\text{th}}$ hidden layer as $\beta_i$. The details are illustrated below.

$$\begin{aligned} \beta_1 &= concat(\underbrace{I^1_1, ..., I^1_{n_1}}_{1^{\text{st}}-order}, ..., \underbrace{I^p_1, ..., I^p_{n_p}}_{p^{\text{th}}-order}, ..., \underbrace{I^q_1, ..., I^q_{n_q}}_{q^{\text{th}}-order}) \\ \beta_z &= relu(W_{z-1}\beta_{z-1} + b_{z-1}) \text{ for } z > 1 \end{aligned} \tag{10}$$

where $W_{z-1}$ and $b_{z-1}$ are the weights and biases for the $(z-1)^{\text{th}}$ layer. The prediction after the *MLP Stage* is

$$\hat{y} = \sigma(W_h \beta_h + b_h) \tag{11}$$

where $h$ is the number of hidden layers, and $\sigma$ is an activation function (normally, the sigmoid function is applied as $\sigma$ in CTR prediction tasks). The cross-entropy loss is used as the objective function:

$$L(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y}) \tag{12}$$

with $y$ as the label.

## 3.6 Optimization

In AutoGroup, we parameterize the selection of different feature sets as a structural optimization problem and make it differentiable by introducing the Gumbel-Softmax trick. The parameters that need optimizing in AutoGroup include: structural parameters $\{\alpha^p_{i,j}\}$ and network weights $\{e_i, W_z, b_z\}$ (embedding parameters and MLP parameters). Structural parameters and network weights can not be optimized simultaneously, as the optimization of one kind is greatly dependent on the other. In DARTs [20], the network weights and the structural parameters are optimized alternatively between gradient

descent steps. More specifically, the training data is split into two parts, such that network weights are firstly updated by the loss on the first part of training data and the structural parameters are updated by the loss on the second part in the next step. However, in our model, we find that satisfactory performance is difficult to achieve by using different parts of training data to optimize network weights and structural parameters. One possible reason is that structural parameters and network weights are essentially competing data for optimization which can be mitigated if there more samples such that different parts of the training data more accurately reflect the true underlying distribution. Hence, instead of using partial training data to optimize structure parameters and network weights respectively, we use the whole training data to optimize them both. More specifically, during the training process, we first generate one batch of data $X_i$, $Y_i$ where $X_i$ is the input data and $Y_i$ is the label for $X_i$. And then $X_i$ and $Y_i$ are used to update network weights. After that, the same $X_i$ and $Y_i$ are used to optimize structure parameters. The above optimization strategy works well in our problem.

## 3.7 Complexity Analysis

*3.7.1 Time Complexity.* In the $p^{\text{th}}$-order of the *Automatic Feature Grouping Stage*, the inference time of feature sets is $O(mn_p)$. Then, computing representation of the feature sets needs $O(mn_pk)$. In the *Interaction Stage*, the time complexity for $p^{\text{th}}$-order interaction in a feature set is $O(pk)$. Hence the time complexity for all feature sets is $O(pn_pk)$. Therefore, the total time complexity before the MLP is $O(\sum_{p=1}^{q} mn_p + mn_pk + pn_pk) = O(\sum_{p=1}^{q} n_pk(m+p))$. After the *Interaction stage*, we will feed the interaction results to the MLP. The dimension of interaction results is $n_1k + \sum_{p=2}^{q} n_p$ (see Eq. 9). In the *MLP Stage*, the time complexity for the first layer is $O((n_1k + \sum_{p=2}^{q} n_p)H_1)$; the time complexity is $H_{z-1}H_z$ for a subsequent $z^{\text{th}}$ layer. To summarize, the total time complexity is

$$O\left(\sum_{p=1}^{q} n_pk(m+p) + (n_1k + \sum_{p=2}^{q} n_p)H_1 + \sum_{z=2}^{h} H_{z-1}H_z\right). \quad (13)$$

*3.7.2 Space Complexity.* In the embedding layer, to avoid the inconsistency when updating parameters, we use different embeddings to represent features for different orders. Hence the space complexity for the embedding layer is $O(qnk)$. In the *Automatic Feature Grouping Stage*, the space complexity is $O(\sum_{p=1}^{q} mn_p)$. In the *Interaction Stage*, there are no additional parameters. In the *MLP Stage* (recall that the input dimension of the first layer is $O(n_1k + \sum_{p=2}^{q} n_p)$), the space complexity is $O((n_1k + \sum_{p=2}^{q} n_p)H_1 + \sum_{z=2}^{h} H_{z-1}H_z)$. To summarize, the total space complexity for AutoGroup is

$$O\left(qnk + \sum_{p=1}^{q} mn_p + (n_1k + \sum_{p=2}^{q} n_p)H_1 + \sum_{z=2}^{h} H_{z-1}H_z\right). \quad (14)$$

# 4 OFFLINE EXPERIMENTS

Extensive offline experiments are conducted on three well-known public datasets to answer the following research questions:

- **RQ1**: How does AutoGroup perform compared with the state-of-the-art CTR prediction models?
- **RQ2**: How effective is each stage of AutoGroup?
- **RQ3**: How efficient is AutoGroup as compared with the state-of-the-art methods?
- **RQ4**: How do different hyper-parameter settings affect the performance of AutoGroup (i.e., orders of feature interactions, number of feature sets in each order)?

**Table 2: Statistics of Evaluation Datasets**

| Dataset | #instances | #dimension | #fields | positive ratio |
|---|---|---|---|---|
| Criteo | $1 \times 10^8$ | $1 \times 10^6$ | 39 | 0.50 |
| Avazu | $4 \times 10^7$ | $6 \times 10^5$ | 24 | 0.17 |
| iPinYou | $2 \times 10^7$ | $9 \times 10^5$ | 16 | 0.07 |

## 4.1 Experimental Setup

*4.1.1 Datasets.* Experiments are conducted in the following three public benchmark datasets:

- **Criteo**[3]: Criteo contains one month of click logs with billions of data samples. We select "data 6-12" as the training set while selecting "day-13" for evaluation. To counter label imbalance, negative down-sampling is applied to keep the positive ratio roughly at 50%. 13 numerical fields are converted into one-hot features through bucketing, where features in a certain field appearing less than 20 times are set as a dummy feature "other".
- **Avazu**[4]: Avazu was released in the CTR prediction contest on Kaggle. 80% of randomly shuffled data is allotted to training and validation with 20% for testing. Categories with less than 20 times of appearance are removed for dimensionality reduction.
- **iPinYou**[5]: iPinYou was published in the iPinYou RTB Bidding Algorithm Competition, 2013. We utilise seasons 2 and 3 as our dataset.

To make a fair comparison with the state-of-the-art models, we process the data in the three datasets exactly the same as in [18, 27]. Table 2 summarizes the characteristics of the above datasets.

*4.1.2 Baselines.* As detailed in Section 2, we will use the following state-of-the-art methods as baselines: **LR** [15], **GBDT** [4], **FM** [29], **FFM**[6] [14], **FNN** [37], **AFM** [34], **IPNN** [26, 27], **PIN** [27], **DeepFM** [9], **xDeepFM** [16], **FGCNN** [18]. Note that HOFM [2] is not compared with because it is shown to be inferior to AFM in [34].

*4.1.3 Evaluation metrics .* AUC (Area Under Curve) and log loss (cross entropy) are selected as our evaluation metrics.

*4.1.4 Hyper-parameter settings.* Table 3 summarizes the hyper-parameters for each model. For all baselines, the hyper-parameters are set to be the same as in [18, 27][7]. Recall that different embeddings are utilized to represent features for different orders of interactions. In real-world applications, one can lessen or enlarge the embedding size for high-order feature interactions, to reduce the complexity or enhance the capacity of the *Interaction Stage*. Note that the results are not sensitive to the values of $n_p$.

[3] http://labs.criteo.com/downloads/download-terabyte-click-logs/
[4] http://www.kaggle.com/c/avazu-ctr-prediction
[5] http://data.computational-advertising.org
[6] The implementation of libFFM (https://github.com/guestwalk/libffm) was used.
[7] As FGCNN is not evaluated on iPinYou in the orignal paper [18], the hyper-parameters of FGCNN on iPinYou dataset is tuned carefully on validation set by us.

**Table 3: Hyper-parameter Settings**

| Params | Criteo | Avazu | iPinYou |
|---|---|---|---|
| General | bs=2000; lr=1e-3<br>opt=Adam | bs=2000; lr=1e-3<br>opt=Adam | bs=2000; lr=1e-3<br>opt=Adam; l2_e=1e-6 |
| LR | – | – | – |
| GBDT | depth=25; #tree=1300 | depth=18; #tree=1000 | depth=6; #tree=600 |
| FM,AFM,<br>KFM,NIFM | n=20; t=0.01; h=32<br>l2_a=0.1; sub-net=[40,1] | n=40; t=1; h=256;<br>l2_a=0; sub-net=[80,1] | n=20; t=1; h=256<br>l2_a=0.1; sub-net=[40,1] |
| FFM | n=4 | n=4 | n=4 |
| CCPM | n=20<br>kernel=[7 × 256]<br>net=[256 × 3,1] | n=40<br>kernel=[7 × 128]<br>net= [128 × 3,1] | n=20<br>kernel=[7 × 128]<br>net=[128 × 3,1] |
| xDeepFM | n=20<br>net=[400 × 3,1]<br>CIN=[100 × 4] | n=40<br>net=[700 × 5,1]<br>CIN=[100 × 2] | n=20<br>net=[300 × 3,1]<br>CIN=[100 × 4]<br>LN=true |
| FNN,DeepFM,<br>IPNN,KPNN | n=20<br>net=[700 × 5,1]<br>LN=true | n=40<br>net=[500 × 5,1]<br>LN=true | n=20<br>net=[300 × 3,1]<br>LN=true |
| PIN | n=20<br>net=[700 × 5,1]<br>sub-net=[40,5]<br>LN=true | n=40<br>net=[500 × 5,1]<br>sub-net=[40,5]<br>LN=true | n=20<br>net=[300 × 3,1]<br>sub-net=[40,5]<br>LN=true |
| FGCNN | k=20<br>conv=9*1<br>kernel=[38,40,42,44]<br>new=[3,3,3,3]<br>BN=T<br>net=[4096,2048,1] | k=40<br>conv=7*1<br>kernel=[14,16,18,20]<br>new=[3,3,3,3]<br>BN=T<br>net=[4096,2048,<br>1024,512,1] | k=20<br>conv=3*1<br>kernel=[4, 6, 8]<br>new=[1,1,1]<br>BN=T<br>net=[600 × 3,1] |
| AutoGroup | k=20<br>lr_h=10<br>$\tau$=0.1<br>net=[1024,512,256]<br>n_p=[35,390,300,<br>500,450,150]<br>high =[20,100,40,80]<br>q=6 | k=40<br>lr_h = 1e4<br>$\tau$ = 0.01<br>net=[1024,512,256,1]<br>n_p=[15,130,170,<br>210,250,290]<br>high=[100,80,100,60]<br>q=6 | k=20<br>lr_h = 1e3<br>$\tau$ = 0.01<br>net=[300 ×3, 1]<br>n_p=[14,120,160,<br>200,240]<br>high =[20,20,20,20]<br>q=5 |

* Note: bs= batch size, opt= optimizer, lr= learning rate, l2_e= $l_2$ regularisation on embedding layer, t= softmax temperature, l2_a= $l_2$ regularisation on attention network, h= attention network hidden size, n= embedding size, net= MLP structure, sub-net= micro network, LN= layer normalization, BN= batch normalization, lr_h= learning rate for structural parameters, $\tau$ = Gumbel-softmax temperature, q= maximum order for explicit interactions, n_p= number of feature sets in each order, high = vector size for high order's (order>2) embedding to increase model's capacity.

*4.1.5 Significance Test.* The experiments for AutoGroup and the best baseline model are repeated 10 times by changing the random seeds. The two-tailed unpaired $t$-test is performed to detect significant differences between AutoGroup and the best baseline.

## 4.2 Overall Performance (RQ1)

Table 6 summarizes the performance of all compared methods on three large-scale public datasets. AutoGroup achieves significant improvement over all baselines in all datasets in both AUC and log loss. We have the following observations:

Firstly, neural network models outperform all non-neural network models, implying that deep neural networks can learn effective feature interactions and make better predictions than linear models (i.e., LR), tree-based models (i.e., GBDT) and FM variants (i.e., FM, FFM, AFM).

Secondly, comparing IPNN and PIN with FNN, we find that explicitly modelling low-order interactions can boost the performance of an MLP. The reason is that although an MLP is able to model arbitrary feature interactions, it requires much more parameters than tensor factorization models, which makes it difficult to train. As a complementary part, explicit low-order feature interactions simplify the training of MLP, so better performance is achieved.

Thirdly, xDeepFM, which explicitly models high-order feature interactions, performs even worse than IPNN and PIN (which both model just $2^{\text{nd}}$-order interactions). It is consistent with our statement: enumerating all feature interactions in a brute-force way and relying on the neural network to identify the important ones among all (as xDeepFM make neural network distracted by the useless interactions and it leads to inferior performance.

Fourthly, FGCNN is the best baseline model on Criteo and Avazu, which combines a CNN and MLP to generate global feature interactions as new features to augment the raw feature space. Its performance shows that implicit high-order interactions can also be used to augment feature space. However, augmenting the feature space also make the MLP more complicated and prone to overfitting. The performance of FGCNN on iPinyou dataset is even inferior to IPNN (which is the simplified version of FGCNN without feature augmentation), because iPinYou dataset, which has less data than Criteo and Avazu, provides insufficient information for FGCNN to be trained well.

Finally, AutoGroup consistently achieves the best performance on all three datasets. Comparing with IPNN and PIN, AutoGroup explicitly models high-order feature interactions before the MLP, which confirms the advantage of modelling high-order interaction explicitly on the prediction performance. Comparing with xDeepFM and FGCNN, AutoGroup automatically identifies important high-order interactions through structural optimization by AutoML techniques, which reduces the challenge for the MLP to find effective feature interactions from the large combination space. Moreover, we have designed a novel interaction function (in Eq. 9) to estimate the $p^{\text{th}}$-order interaction in a selected feature group. AutoGroup, having many fewer parameters than xDeepFM and FGCNN, is easier to learn. The superior performance of AutoGroup over xDeepFM and FGCNN validates the effectiveness of our method for modelling high-order feature interactions explicitly.

## 4.3 Online Experiments

We further conduct online experiments in a mainstream app recommender system to test the real-world performance of AutoGroup. Specifically, a ten-day A/B test is carried out in a game recommendation scenario in the mainstream App Store. For the control group, 5% of users are randomly selected and presented with recommendations generated by the baseline, which is a fine-tuned DeepFM model [9]. DeepFM is chosen as a strong baseline due to its extraordinary accuracy and high efficiency, which has been deployed in our commercial system. For the experimental group, 5% of users are presented with recommendations generated by AutoGroup. Since there are tens of millions of active users visiting our App Store daily, the conclusion of the online experiments is convincing.

We adopt two metrics to compare the online performance of AutoGroup and DeepFM, namely Click-Through Rate: $CTR = \frac{\#downloads}{\#impressions}$ and Conversion Rate: $CVR = \frac{\#downloads}{\#users}$, where $\#downloads$, $\#impressions$ and $\#users$ are the number of downloads, impressions and visited users in one day of the A/B test period, respectively.

For efficiency consideration, we limit the maximum order of interactions in AutoGroup to 3. The metrics of CTR and CVR are both consistently improved by **10%** (with a p-value < $1e^{-5}$) on average across the ten-day A/B test, which validates the superiority of AutoGroup. Besides accuracy comparison, we also record the inference time of AutoGroup and DeepFM. In our production environment, AutoGroup needs 35% extra inference time over DeepFM, which is practically acceptable.

To summarize, AutoGroup achieves a consistent improvement of 10% in CTR and CVR over DeepFM in a ten-day A/B test with

**Table 4: Performance comparison of different models. The column RI denotes the average relative improvement for AUC and log loss on AutoGroup over Baselines. $^{\star\star}$ and $^{\star}$ represents significance level $p$-value< $10^{-7}$ and $p$-value <0.05 of comparing AutoGroup with the best baseline (indicated by underlined numbers).**

| Model | Criteo | | | Avazu | | | iPinYou | | |
|---|---|---|---|---|---|---|---|---|---|
| | AUC | log loss | RI | AUC | log loss | RI | AUC | log loss | RI |
| LR | 78.00% | 0.5631 | +3.65% | 76.76% | 0.3868 | +3.35% | 76.38% | 0.005691 | +2.88% |
| GBDT | 78.62% | 0.5560 | +2.64% | 77.53% | 0.3824 | +2.29% | 76.90% | 0.005578 | +1.55% |
| FM | 79.09% | 0.5500 | +1.81% | 77.93% | 0.3805 | +1.78% | 77.17% | 0.005595 | +1.52% |
| FFM | 79.80% | 0.5438 | +0.80% | 78.31% | 0.3781 | +1.22% | 76.18% | 0.005695 | +3.05% |
| AFM | 79.13% | 0.5517 | +1.93% | 78.06% | 0.3794 | +1.55% | 77.71% | 0.005562 | +0.87% |
| FNN | 79.87% | 0.5428 | +0.66% | 78.30% | 0.3778 | +1.19% | 77.82% | 0.005573 | +0.90% |
| DeepFM | 79.91% | 0.5423 | +0.59% | 78.36% | 0.3777 | +1.14% | 77.92% | 0.005588 | +0.97% |
| IPNN | 80.13% | 0.5399 | +0.23% | 78.68% | 0.3757 | +0.67% | 78.17% | 0.005549 | +0.46% |
| PIN | 80.18% | 0.5394 | +0.16% | 78.72% | 0.3755 | +0.62% | 78.22% | 0.005547 | +0.41% |
| xDeepFM | 80.06% | 0.5408 | +0.36% | 78.55% | 0.3766 | +0.87% | 78.04% | 0.005555 | +0.60% |
| FGCNN | 80.22% | 0.5389 | +0.08% | 78.82% | 0.3747 | +0.45% | 77.85% | 0.005612 | +1.22% |
| AutoGroup | **80.28%**$^{\star\star}$ | **0.5384**$^{\star\star}$ | - | **79.15%**$^{\star\star}$ | **0.3729**$^{\star\star}$ | - | **78.59%**$^{\star}$ | **0.005528**$^{\star}$ | - |

acceptable inference overhead. The improvement can contribute to significant revenue growth in real-world applications.

## 4.4 Ablation Study (RQ2)

We conduct detailed experiments to answer why AutoGroup outperforms the state-of-the-art models. Firstly, we study the maximum order of explicit feature interactions that AutoGroup models. Then, different AutoGroup variants are implemented and compared to validate the design of the *Automatic Feature Grouping Stage* and *Interaction Stage*.

*4.4.1 Maximum Order Analysis.* To study the effect of the maximum interaction order on AutoGroup's performance, we vary it from 2 to 6. Table 5 summarizes the performance on Criteo and Avazu.

As higher-order interactions are explicitly modelled in AutoGroup, AUC and log loss are gradually improved, which verify that explicit high-order interactions are beneficial to current models and AutoGroup can model them effectively. Moreover, as the maximum order increases, performance improvement gradually decreases. This is a reasonable phenomenon because the higher the order is, the sparser the useful feature interactions are in the combination space, making it difficult to identify them.

Meanwhile, AUC and log loss of AutoGroup have significant improvement over the previous order when the maximum order is 4 and 5 on Criteo (the maximum order is 3, 4 and 5 on Avazu). It shows that different datasets prefer different interaction orders in real-world applications.

AutoGroup starts to outperform FGCNN (which is the best baseline) when the maximum order is 4 on Criteo and 3 on Avazu, demonstrating the effectiveness of AutoGroup in identifying important high-order interactions.

*4.4.2 Study on AutoGroup Variants.* In AutoGroup, there are two functional stages: the *Automatic Feature Grouping Stage* and *Interaction Stage*. In the first stage, we automatically group features into different feature sets, such that feature interaction in each set is effective, in an automatic data-driven manner by AutoML techniques. In the *Interaction Stage*, inspired by the reformulation of Factorization Machine, we define an interaction function (in Eq. 9) for features in the same set to interact at a given order. To validate the effectiveness of these two stages, we propose and compare the following four variants:

**Table 5: Performance of AutoGroup with different maximum interaction orders. $^{\star\star}$ and $^{\star}$ denote the significance level $p < 10^{-4}$ and $p < 0.05$ when comparing the performance of the current maximum order with the previous maximum order. $^{\bullet}$ means AutoGroup with this maximum order outperforms the best baseline (FGCNN) significantly.**

| Max order | Criteo | | Avazu | |
|---|---|---|---|---|
| | AUC | log loss | AUC | log loss |
| FGCNN | 80.22% | 0.5389 | 78.82% | 0.3747 |
| 2 | 80.19% | 0.5292% | 78.83% | 0.3749 |
| 3 | 80.20% | 0.5391 | 79.05%$^{\star\star\bullet}$ | 0.3734$^{\star\star\bullet}$ |
| 4 | 80.26%$^{\star\star\bullet}$ | 0.5386$^{\star\star\bullet}$ | 79.09%$^{\star}$ | 0.3732$^{\star}$ |
| 5 | 80.27%$^{\star}$ | 0.5383 | 79.14%$^{\star}$ | 0.3729$^{\star}$ |
| 6 | 80.28% | 0.5382 | 79.15% | 0.3729 |

**Table 6: Average performance of AutoGroup variants in $6^{\text{th}}$-order with 20 runs.**

| Model | Criteo | | Avazu | |
|---|---|---|---|---|
| | AUC | log loss | AUC | log loss |
| Random | 80.27% ± 0.01% | 0.5385 ± 0.0003 | 79.03% ± 0.04% | 0.3736 ± 0.0003 |
| All | 80.25% ± 0.01% | 0.5386 ± 0.0003 | 79.02% ± 0.03% | 0.3737 ± 0.0002 |
| KeepReg | 80.27% ± 0.01% | 0.5384 ± 0.0001 | 79.01% ± 0.04% | 0.3738 ± 0.0003 |
| AutoGroup | 80.28% ± 0.01% | 0.5384 ± 0.0001 | 79.08% ± 0.03% | 0.3733 ± 0.0002 |

- **Random**: Features in each feature set are randomly selected and such selections are then fixed during the training.
- **All**: Each feature set contains all features to generate feature interaction at a given order, which is similar to xDeepFM [16].
- **KeepReg**: In this variant, the term of $\sum_{f_i \in s_j^p} (w_i^p e_i)^p$ in Eq. 9 is not subtracted, which can be viewed as keeping a $L_p$-norm regularization on the embeddings in the interaction function.

Note that the variants **Random**, and **All** are performed to study the effectiveness of feature set selection in the *Automatic Feature Grouping Stage*; while the variant **KeepReg** is proposed to validate the interaction function in the *Interaction Stage*.

These variants are running with the same hyper-parameter setting as AutoGroup, and we repeat the experiments 20 times for each. Table 6 summarizes the performance of different variants on Criteo and Avazu. We have the following observations:

- The performance of **Random** is better than that of **All** on two datasets. Since **All** contains all high order interactions in each feature set of each order, it has many repeated and

useless interactions in each order, making it difficult to identify useful interactions. However, **Random** randomly drops some interactions in each feature set, which can alleviate the learning difficulties brought by the repeated and useless interactions.

- AutoGroup outperforms **Random**, which shows the effectiveness of using AutoML techniques to learn to group important features in the *Automatic Feature Grouping Stage* through a data-driven manner.
- **KeepReg** is inferior to AutoGroup, which indicates that the normalization terms in the interaction function have negative effects on the model performance. In current state-of-the-art models such as xDeepFM, such normalization terms are not excluded, which can be one reason that it does not work as expected.

To summarize, we show that sparse grouping **(Random)** is always better than non-sparse grouping**All**. And in the *Automatic Feature Grouping Stage*, AutoGroup utilizes AutoML techniques to identify important features and group them to interact which performs better than **Random Grouping**. In the interaction Stage, the side effect of normalization terms is demonstrated.

## 4.5 Efficiency Analysis (RQ3)

In real-world application scenarios, both the effectiveness and efficiency of a model are critical. Actually, many state-of-the-art models achieve good accuracy but fail to be deployed in commercial recommender systems due to their inefficiency in both training and inference. In this section, we study the efficiency of AutoGroup, by comparing its training and inference time with PIN and FGCNN (which are the best baselines) on 100 batches of data (with batch size 2,000) on one NVIDIA P100 GPU.

Figure 2 presents the training and inference time (which is scaled by $ln(1+x)$) of AutoGroup and baseline models. We can observe that AutoGroup needs less training time than FGCNN but achieves better performance on Criteo (with AG(4), AG(5) and AG(6)) and Avazu (with AG(2) and AG(3)).

Inference time is more crucial than training time when deploying a model in industrial applications, as there is always a hard constraint on inference latency there. As can be seen, the inference time for AG(6) is close to PIN, which is only a third of FGCNN.

To summarize, AutoGroup can achieve better performance more efficiently, which makes it easier to be deployed in industrial applications. Meanwhile, it is very flexible to make a trade-off between performance and complexity of AutoGroup by tuning the maximum order of feature interactions.

## 4.6 Hyper-parameter Study (RQ4)

There are some important hyper-parameters in AutoGroup. Due to the space limit, only validation for important hyper-parameters is provided. Since the maximum order of explicit feature interactions has been studied in the previous section, we study the number of feature sets in each order one in this section. AutoGroup selects multiple feature sets for each order of feature interactions to achieve better generalizability, which is inspired by Random Forest. Figure 3 presents the performance of AutoGroup as the number of feature sets varies. For simplicity, we study the case when the maximum

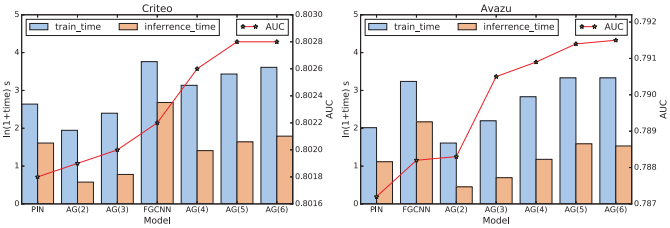


**Figure 2: Training and inference time for 100 batches (batch size 2,000) of different models ordered by their AUC performance where AG($q$) denotes AutoGroup with the maximum order of interaction being $q$.**

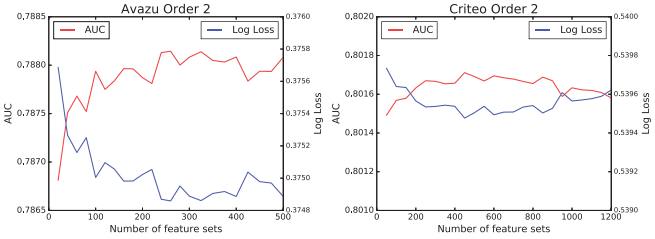


**Figure 3: Study on the number of feature sets at the $2^{nd}$-order interactions on Avazu and Criteo.**

order of feature interactions is set to 2. As the figure shows, when the number of feature sets starts to increase, the performance is improved, because more feature sets are able to capture more useful feature interactions. However, when the feature sets saturate, the model performance stops to benefit from including more sets, as it is enough to identify important interactions and more feature sets may introduce more useless feature interactions. Figure 3 shows that AutoGroup achieves the best performance when the number of feature sets for $2^{nd}$-order interaction is 240 and 450 on Avazu and Criteo respectively. We also briefly note that the hyper-parameter $\tau$ only affects AUC by 0.01% on Criteo.

## 5 CONCLUSION

In this paper, we propose a novel three-stage model, AutoGroup, to automatically identify and select useful high-order feature interactions. To achieve this, the *Automatic Feature Grouping Stage* selects multiple feature sets for each order, such that the $p^{\text{th}}$-order interaction for each feature set is effective in boosting the prediction performance. The selection process of high-order feature interactions is modelled as a structural optimization problem which is solved efficiently by AutoML techniques. In the *Interaction Stage*, a novel interaction function on the representation of a feature set is proposed to produce feature interactions for a given order. Finally, in the *MLP Stage*, interactions of all orders are fed into an MLP.

Extensive offline experiments on three large-scale public benchmark datasets demonstrate the superior performance of AutoGroup. A detailed ablation study elaborates on the effectiveness of each stage in AutoGroup. Comparison of training and inference time between AutoGroup and the baselines shows its advantage in efficiency, which makes it practical to be deployed in real-world industrial systems. We implement and deploy AutoGroup in a mainstream online app recommender system. A ten-day A/B test shows 10% improvements in terms of CTR and CVR over a strong baseline.

## REFERENCES


[1] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 46–54.

[2] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. 2016. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*. 3351–3359.

[3] Yin Wen Chang, Cho Jui Hsieh, Kai Wei Chang, Michael Ringgaard, and Chih Jen Lin. 2010. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. *Journal of Machine Learning Research* 11, 11 (2010), 1471–1490.

[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost:A Scalable Tree Boosting System. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[5] Hengtze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Deepak Chandra, Hrishi Aradhye, Glen Anderson, Greg S Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & Deep Learning for Recommender Systems. *conference on recommender systems* (2016), 7–10.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247* (2017).

[10] Xiangnan He and TatSeng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. (2017), 355–364.

[11] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, and Stuart Bowers. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Eighth International Workshop on Data Mining for Online Advertising*. 1–9.

[12] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[13] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[14] Yuchin Juan, Yong Zhuang, Wei Sheng Chin, and Chih Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *ACM Conference on Recommender Systems*. 43–50.

[15] Kuang Chih Lee, Burkay Orten, Ali Dasdan, and Wentong Li. 2012. Estimating conversion rate in display advertising from past erformance data. In *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*. 768–776.

[16] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *arXiv preprint arXiv:1803.05170* (2018).

[17] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[18] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. www2019. (2019). https://doi.org/10.1145/3308558.3313497 arXiv:arXiv:1904.04447

[19] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[21] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Advances in neural information processing systems*. 7816–7827.

[22] Chris J Maddison, Daniel Tarlow, and Tom Minka. 2014. A* sampling. In *Advances in Neural Information Processing Systems*. 3086–3094.

[23] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[24] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 1349–1357.

[25] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[26] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.

[27] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based Neural Networks for User Response Prediction over Multi-field Categorical Data. *arXiv preprint arXiv:1807.00311* (2018).

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4780–4789.

[29] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[30] Steffen Rendle. 2012. Factorization Machines with LibFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article Article 57 (May 2012), 22 pages.

[31] Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, and Guangzhong Sun. 2017. Practical Lessons for Job Recommendations in the Cold-Start Scenario. In *Recommender Systems Challenge*. 4.

[32] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[33] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[34] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[35] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[36] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data. (2016).

[37] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data: A Case Study on User Response Prediction. (2016).

[38] Xinbang Zhang, Zehao Huang, and Naiyan Wang. 2018. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567* (2018).

[39] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5941–5948.

[40] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1059–1068.