A General Method For Automatic Discovery of Powerful Interactions In Click-Through Rate Prediction

Ze Meng1,∗†, Jinnian Zhang2,∗†, Yumeng Li3, Jiancheng Li3, Tanchao Zhu4, Lifeng Sun4,5,†
1Beijing Key Lab of Networked Multimedia, Dept. of Computer Science and Technology, Tsinghua University, China
2Dept. of Electrical and Computer Engineering, University of Wisconsin Madison, USA
3Alibaba Group, China
4Key Laboratory of Pervasive Computing, Tsinghua University, Ministry of Education, China
5BNRist, Dept. of Computer Science and Technology, Tsinghua University, China
†mengz18@tsinghua.org.cn, jinnian.zhang@wisc.edu, sunlf@tsinghua.edu.cn

Abstract
Modeling powerful interactions is a critical challenge in Click-through rate (CTR) prediction, which is one of the most typical machine learning tasks in personalized advertising and recommender systems. Although developing hand-crafted interactions is effective for a small number of datasets, it generally requires laborious and tedious architecture engineering for extensive scenarios. In recent years, several neural architecture search (NAS) methods have been proposed for designing interactions automatically. However, existing methods only explore limited types and connections of operators for interaction generation, leading to low generalization ability. To address these problems, we propose a more general automated method for building powerful interactions named AutoPI. The main contributions of this paper are as follows: AutoPI adopts a more general search space in which the computational graph is generalized from existing network connections, and the interactive operators in the edges of the graph are extracted from representative hand-crafted works. It allows searching for various powerful feature interactions to produce higher AUC and lower Logloss in a wide variety of applications. Besides, AutoPI utilizes a gradient-based search strategy for exploration with a significantly low computational cost. Experimentally, we evaluate AutoPI on a diverse suite of benchmark datasets, demonstrating the generalization and efficiency of AutoPI over hand-crafted architectures and state-of-the-art NAS algorithms.

CCS Concepts
• Information systems → Recommender systems; • Computing methodologies → Neural networks; • Mathematics of computing → Nonconvex optimization.

Keywords
Click-through Rate Prediction, Gradient-based Neural Architecture Search, Feature Interaction, Interaction Ensemble

1 Introduction
Predicting the user’s clicking probability on a given item is a crucial task in a recommender system. Essentially, the key challenge in CTR prediction is how to model powerful feature interactions effectively. The classical methods employ a single type of interactive operator to model the fixed-order feature interactions in an explicit manner. For example, Factorization Machines (FM) based methods [25, 30] learn 2nd-order cross features using inner product. And its variant, High-order Factorization Machine (HOFM) [1], adopted Analysis of Variance (ANOVA) to approximate high-order cross features, while only marginal improvement was observed despite using more parameters. Deep learning introduces a more effective operator, Single-layer Perceptron (SLP) [9], which can be stacked to form Multi-layer Perceptron (MLP) to generate high-order feature interactions in an implicit manner. However, the representation ability of the cross features generated from a single type of operator is limited. As an enhancement to MLP, incorporating explicit lower-order feature interactions in deep learning models achieves better performance, namely, multi-interaction ensemble, such as Wide&Deep [2], DeepFM[7] and xDeepFM [14].

However, manually developing the interactions for generalizing across diverse scenarios is computationally intensive and is a black box optimization without an optimality guarantee. Instead of fixing the order of interactions, learning the order of interactions or adaptively identifying important groups of features within a predefined order attract increasing attention. Specifically, by using the multi-head self-attention [27], the relevant features can be grouped to form meaningful arbitrary-order cross features. AFN [3] learns adaptive-order cross features via a logarithmic transformation layer. And more advanced NAS methods, AutoFIS [18] automatically identify important cross features from all possible 2nd-order and 3rd-order combinations of features with inner product. As a generalization of AutoFIS, AutoGroup [17] can generate substantial interactions of any order in an automatic manner. AutoCross [20] explicitly search for useful high-order cross features in a tree-based search space by an efficient beam search method. However, the above methods generate the interactions by using only a single type of operator for all identified groups of features. In order to enable the automatic search for the best connections of operators for generating the interaction from each group, AutoFeature [12] encodes the subnetwork for each candidate group of features within a predefined order as a string. Note that each subnetwork can be considered as a direct acyclic graph (DAG) of operators. Then a tree of Naïve Bayes classifiers is used for optimization. AutoCTR [26] defines three basic blocks from operators SLP, FM, and dot product (DP), and utilize the evolutionary algorithm to obtain the best connections of these blocks. Similarly, in AutoRec [29], five different blocks are created based on several operators such as SLP, FM, outer product, self-attention, etc. Three strategies (random,
Nevertheless, these works only involve simple operators and limited possible connection types for building the interactions, leading to narrow search space. For generalization to a large number of scenes, we study a more general paradigm of discovering powerful interactions. We search the interactions of CTR prediction according to three key factors: (1) the order of interaction in each tower, (2) the types and connections of operators used to construct interactions, (3) the multi-tower structure for the ensemble of interactions. Specifically, we review various representative literature to extract and modulize significant operators to form the operation space and design the tailored cell-based computational graph containing more possible connections of operators. Each node in the graph represents a latent representation (e.g., intermediate cross features), and the edge is an operator from the operation space. Notably, we unify the concept of the order related to explicit and implicit interactions following [14]. For example, DeepFM contains two types of interactive operators, FM and SLP. FM explicitly produces 2nd-order cross features, while MLP implicitly generates 4th-order cross features because it has three SLPs. Thus, our method discovers powerful interactions by integrating multiple interactions, where each interaction consists of different numbers, types, and connections of operators. However, our search space exponentially increases with the aforementioned three key factors of the interactions, requiring an efficient search strategy. Inspired by DARTS [19], we describe an improved gradient-based method for exploration to improve efficiency. In detail, we introduce architecture parameters to relax the discrete set of operators and node connections, and utilize the Bi-level optimization algorithm to optimize the architecture and weights by gradient descent. Moreover, two different training techniques are proposed to alleviate a well-known problem in the weight-sharing mechanism, w.r.t. the performance gap [31].

To summarize, the main contributions of this paper can be highlighted as follows:

- Designing a general and efficient search space to generate powerful interactions, which consists of the operation space and the computational graph. The operation space is constructed via extracting and modulizing the representative interactive operators in existing literature, and we design a tailored cell-based computational graph for finding the best connections of the operators.
- A gradient-based search strategy is applied for exploration in our broad search space to improve efficiency. We introduce the architecture parameters for relaxing the discrete search space and use a Bi-level optimization algorithm to optimize the architecture and weights iteratively.
- Two training techniques are utilized for alleviating the performance gap between the continuous architecture encoding and the derived architecture.
- Extensive experiments demonstrate the generalization and efficiency of AutoPI on datasets in different scales. Comparing with SOTA methods, on average, our method increases the AUC by 0.46% and 0.70% on the public and private datasets, respectively. Furthermore, the gradient-based search strategy can reduce 90% of the search overhead compared to the NAS baselines.

Our work is organized as follows. In Section 2, we summarize the related work on NAS and CTR prediction. Section 3 formally defines our problem. Section 4 presents the details of our method. In Section 5, we present the experimental results and detailed analysis. Finally, we conclude this paper and point out the future work in Section 6.

2 Related Work

Click-Through Rate Prediction. Traditional models aim to formulate the explicit fixed-order interactions for CTR prediction, such as LR, FM, Attention Factorization Machine (AFM), HOFM, and CrossNet [22]. Different from HOFM, the number of parameters in CrossNet only increases linearly with the input dimension.

Compared to polynomial interactions in FM, CrossNet, etc., complicated non-linear features created through neural networks [3, 8, 22, 27] can be more informative and thus improve the prediction performance. PNN [22] builds the inter-field interactions by using the inner product or outer product and feeds it to an MLP. NFM [8] has a similar architecture as PNN, but uses a Bi-Interaction layer to model the inter-field interactions. Instead of interactions generated by a single model, multi-interaction ensembles are known as the boost of performance by exploiting the advantages of different models. In Wide&Deep [2], it combines LR and Deep Neural Network (DNN), which outperforms each individual model. DeepFM [7] comprises DNN and FM, and other ensemble models include Deep&Cross [28] and xDeepFM [14]. These models have been verified to be effective on many public datasets. In recent years, some hand-crafted models try to learn the order of feature interactions. For example, AutoInt [27] makes use of the latest techniques, attention and residual networks, to generate non-linear features. AFN [3] learns the powers (orders) of each feature in interactions by a logarithmic transformation layer.

Neural Architecture Search. The primary motivation for NAS is to automate the laborious process of designing neural networks in different tasks. There are basically four existing frameworks for NAS [6]: evolution-algorithm based NAS [23], reinforcement-learning based NAS [21], Bayesian optimization based NAS [6], and gradient-based NAS [24]. Among various NAS methods, DARTS [19] and its variants [4, 5, 13, 15, 32] have become the most popular algorithms due to remarkable efficiency improvement in time cost. The core of DARTS is to utilize continuous relaxation of the architecture representation and apply Bi-level optimization to iteratively update the architecture and its weights by using gradient descent. The effectiveness of DARTS has been proved in both Computer Vision (CV) and Neural Language Processing (NLP) domains. Besides existing literature on NAS methods are focused on the tasks in CV and NLP, there is an increasing interest in applying NAS to recommender systems in recent years, such as AutoCTR, AutoFeature and AutoRec, etc. These methods are demonstrated to yield higher performance than human-crafted models.

3 Problem Definition

We summarize the general framework of CTR prediction in Figure 1. The deep CTR prediction models have three fundamental stages from a bottom-up perspective: (i) input transformation, (ii) modeling diverse feature interactions, and (iii) multi-interaction ensemble.
We formally introduce the definitions of three subproblems in CTR prediction as follows:

**DEFINITION 1. (Input Transformation)** We represent the raw feature vector as \( \mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_m] \), where \( m \) is the number of total fields, \( \mathbf{x}_i \) is a one-hot feature representation of the \( i \)-th field, which is categorical (e.g., gender=male, name=Alan, age=25). Since the feature representations are very sparse and high-dimensional, we employ embedding layers to convert the sparse feature into a low dimensional and dense real-value vector \( \mathbf{e}_i = \mathbf{V}_i \mathbf{x}_i \), where \( \mathbf{V}_i \) is an embedding matrix for field \( i \). Then, the output of the embedding layer will be a feature matrix \( \mathbf{E} \in \mathbb{R}^{m \times k} \) by stacking multiple embedding vectors \( [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m] \), where \( k \) is the embedding size of each field. To avoid interference between the deep and shallow model [3], we adopt dual-embedding\(^1\) (i.e., two independent embedding layers \( \mathbf{V}^{low} \) and \( \mathbf{V}^{high} \)) to build lower-order and higher-order feature interactions, respectively.

**DEFINITION 2. (Feature Interactions)** The key problem is to determine which types and orders of interactions should be built to form meaningful cross features. We first redefine the order of interactions from the convention following [14]. Mathematically, Given an input feature matrix \( \mathbf{E} \), a \( p \)-order cross feature is defined as:

\[
\mathbf{E}^p = o^{(p-1)}(o^{(p-2)}(\ldots(o^{(1)}(\mathbf{E})\ldots))).
\]

(1)

Namely, the \( p \)-order cross feature \( \mathbf{E}^p \) is generated by the composition of \( p-1 \) operators \( o^{(1)}(\cdot), o^{(2)}(\cdot), \ldots, o^{(p-1)}(\cdot) \) in order, such as FM and SLP. Traditionally, this is designed by domain experts based on their knowledge. In this paper, we tackle this problem with a novel method, i.e., NAS.

**DEFINITION 3. (Multi-interaction Ensemble)** In hand-crafted CTR prediction models, one of the most critical designs is the multi-interaction ensemble, which combines various orders and types of interactions to form a multi-tower structure. For final prediction, the cross features of all towers are concatenated and then delivered to an SLP as follows:

\[
\hat{y} = \sigma \left( \mathbf{w}^T \left( \mathbf{e}^{(1)} \oplus \mathbf{e}^{(2)} \oplus \ldots \oplus \mathbf{e}^{(n)} \right) + b \right),
\]

(2)

where \( \mathbf{w} \) is a column projection vector which linearly combines concatenated features, and \( b \) is the bias. Symbol \( \oplus \) is the concatenation operation, and \( \sigma(x) = 1/(1 + e^{-x}) \) transforms the logits to users’ clicking probabilities. Note that \( \mathbf{e}^{(i)} \) is the flattened cross features of \( \mathbf{E}^{(i)} \) from the \( i \)-th tower, and \( n \) is a hyperparameter that indicates the total number of towers. The Logloss is adopted for training the model, which is defined as follows:

\[
\text{Logloss} = -\frac{1}{N} \sum_{i=1}^{N} \left( y_i \log (\hat{y}_i) + (1 - y_i) \log (1 - \hat{y}_i) \right),
\]

(3)

where \( y_i \) and \( \hat{y}_i \) are the ground truth of user clicks and predicted CTR, respectively, and \( N \) is the number of training samples.

4 AutoPI

4.1 Method Overview

The work flow of AutoPI is shown in Figure 2. Roughly speaking, our method includes three parts: (i) search space (Section 4.2), (ii) search strategy (Section 4.3) and (iii) performance evaluation (Section 4.4). Specifically, we adopt the cell-based search space, which consists of the computational graph (Section 4.2.1) and the operation space (Section 4.2.2). The efficient gradient-based search strategy is utilized to search for the best connections of operators. Its core is a continuous relaxation (Section 4.3.1) scheme for our discrete search space, leading to a differentiable learning objective for the joint optimization of the architecture and its weights (Section 4.3.2). We also apply the Batch Normalization [11] technique, because of its capability of dealing with numerical instability and improving fairness among operators (Section 4.3.3). After the search process, the discretization technique (Section 4.4.1) is used to obtain the discrete architecture from continuous architecture encoding. For alleviating the performance gap before and after discretization, temperature anneal and noisy skip-connection can be utilized in the search process (Section 4.4.2). The final procedure is training the model weights with the powerful architecture from scratch.

4.2 Search Space Design

We summarize the interactions of hand-crafted models as three critical factors: (1) The orders of interactions, w.r.t, how many operators should be combined to form an interaction. (2) Operators used to construct interactions, w.r.t, which operators should be selected and how operators are connected to build an interaction. (3) Multi-interaction ensemble, w.r.t, which interactions should be included in the final interaction ensemble. In the search space design, we use the cell-based operation space to model the computational graph and the operation space to model the architecture space.
The task of learning the cell, therefore, reduces to choosing the best interaction cell formulates the higher-order feature interactions. Specifically, the interaction cell has two input nodes, where node 5 is the matrix of the higher-order cross features generated from the interaction cell, and node 6 is the input embedding matrix produced by another of dual-embedding layers. Note that the intermediate nodes (node 7, 8) in the ensemble cell function as the output from two towers. The output node 9 is a concatenation of all intermediate nodes. Since the number of intermediate nodes is configurable, the number of towers in the ensemble cell can be adapted to the scenarios. It is notable that although the cells are separate DAGs, instead of optimizing the cells sequentially, a joint optimization can also be applied.

In summary, we explicitly search for higher-order and lower-order interactions, and integrate them by a multi-tower structure. According to our experiments, sharing embedding layers between deep and shallow models leads to a bad performance. Therefore, We adopt the dual-embedding layers to avoid the gradient interference between different cells [3].

4.2.2 Operation Space. To search for various interactions, diverse interactive operators should be included in our search space. We extract and modulize interactive operators with the following considerations [26]:

• Functionality: Operators in our search space should complement and accommodate each other for applications in extensive scenarios. For example, complex operators (e.g., SLP, convolution layer, etc.) have strong learning ability, modeling more complex cross features. In contrast, simple operators (e.g., LR, SENET Layer, etc.) can learn general representation with a low risk of overfitting.

• Operator granularity: The fine-grained operators (e.g., FM, SLP, etc.) have substantial combinational flexibility than coarse-grained (e.g., AFM, MLP, etc.). Because the NAS method is sensitive to computational overhead, efficient operation space requires operators with lower time-space complexity. Thus fine-grained operators are preferable.

• Dimension alignment: In our computation graph, we need to maintain the dimension of the output feature \( o(X) \in \mathbb{R}^{m \times k} \) as the same as the input feature \( X \in \mathbb{R}^{m \times k} \).

According to the above principles, we extract and modify the operators from representative NAS and CTR prediction literature, such as LR, inner product, hadamard product, outer product, cross layer [28], none [19], skip-connection, self-attention [27], FM [25], logarithmic transformation layer [3], SENET layer [10], 2d convolution [16], 1d convolution, etc. After extensive experiments and careful comparison, we finally determine the following operators to form the operation space:

• Skip-connection: takes an input feature matrix \( X \in \mathbb{R}^{m \times k} \) and outputs the same feature matrix \( X' = X \in \mathbb{R}^{m \times k} \), which is an identity mapping and does not increase the order of interactions.
where the weights of operators are parameterized by a vector \( \mathbf{a} = [a_1, a_2, \ldots, a_m] \), and then rescales \( \mathbf{X} \) with vector \( \mathbf{a} \) to obtain a weighted feature matrix \( \mathbf{X}' = [a_1 \cdot \mathbf{x}_1, a_2 \cdot \mathbf{x}_2, \ldots, a_m \cdot \mathbf{x}_m] \in \mathbb{R}^{m \times k} \).

- **Self-attention [27]**: takes an input feature matrix \( \mathbf{X} \in \mathbb{R}^{m \times k} \) and outputs cross features \( \mathbf{X}' \in \mathbb{R}^{m \times k} \) via the key-value attention mechanism.

- **FM**: takes an input feature matrix \( \mathbf{X} \in \mathbb{R}^{m \times k} \) and produces an inner product vector \( \mathbf{p} = \{(\mathbf{x}_i \cdot \mathbf{x}_j)\}_{(i,j) \in \mathcal{R}_e} \), where \( \mathcal{R}_e = \{(i, j) \in \{1, \ldots, m\}, i \neq j\} \), and then outputs the feature matrix \( \mathbf{X}' = \mathbf{p} \cdot \mathbf{W} \in \mathbb{R}^{m \times k} \) via a linear transformation for dimension alignment.

- **Single-layer Perceptron**: takes an input flattened feature vector \( \mathbf{x} \in \mathbb{R}^m \) and outputs a feature matrix \( \mathbf{X}' = \mathbf{x} \cdot \mathbf{W} \in \mathbb{R}^{m \times k} \) via a linear transformation.

- **1d Convolution**: takes an input feature matrix \( \mathbf{X} \in \mathbb{R}^{m \times k} \) and outputs a feature matrix \( \mathbf{X}' \in \mathbb{R}^{m \times k} \) via convolving with \( m \) kernel matrices \( \{\mathbf{C}(i)\}_{i \in \{1, \ldots, m\}} \in \mathbb{R}^{m \times k \times 1} \).

Note that each operator is followed by a nonlinear activation function (e.g. ReLU(\( \cdot \))), excluding Skip-connection.

### 4.3 Search Strategy

The gradient-based search strategy is adopted in our method for computational efficiency. We first introduce architecture parameters for continuous relaxation on discrete search space. By doing so, the architecture can be optimized by the gradient descent method.

#### 4.3.1 Continuous Relaxation

Let \( \mathcal{O} \) be a set of candidate interactive operators, where each operator is denoted by a function \( o(\cdot) \) to be applied to \( \mathbf{X}(i) \). Within a cell, the goal is to choose the best operator from \( \mathcal{O} \) to connect each pair of nodes. Let a pair of nodes be \( (i, j) \), where \( i < j \), the key of continuous relaxation is to convert the combinatorial optimization problem to find the best weights of all operators between \( (i, j) \) to be assigned to a weighted sum over \( \mathcal{O} \) operators, namely,

\[
f_{ij}(\mathbf{X}(i)) = \sum_{o \in \mathcal{O}} \frac{\exp(\sigma_o(i,j)/\tau)}{\sum_{o' \in \mathcal{O}} \exp(\sigma_o(i,j)/\tau)} o(\mathbf{X}(i))
\]

where the weights of operators are parameterized by a vector \( \sigma_{o(i,j)} \) with dimension \( \mathcal{O} \), and \( \tau > 0 \) is a parameter called temperature for controlling the irrelevance of operators across all operators. Besides, operation-level parameters \( a \), we also introduce edge-level parameters \( \beta \) for choosing important node pairs in an interaction cell. And the Equation 4 can be rewritten as:

\[
\mathbf{X}(i) = \sum_{i \leq j \leq i} \frac{\exp(\beta(i,j)/\tau)}{\sum_{i \leq j \leq i} \exp(\beta(i,j)/\tau)} \cdot f_{ij}(\mathbf{X}(i)).
\]

As a result, we omit the operation widely used in most gradient-based NAS methods [4, 13, 15, 19, 32], and normalize \( \mathbf{X}(i) \) for numerical stability. The task of architecture search then reduces to learning two sets of continuous variables \( \mathbf{a} = \{a(\mathbf{i}, \mathbf{j})\}_{i < j}, \beta = \{\beta(\mathbf{i}, \mathbf{j})\}_{i < j} \).

#### 4.3.2 Bi-level Optimization

Following [19], we use the Bi-level algorithm to alternatively learn the architecture parameters \( a, \beta \) and the weights \( w \) of all operators in the architecture. Specifically, the goal for searching powerful interactions is to find \( a^*, \beta^* \) that minimizes the validation loss \( \mathcal{L}_{\text{val}}(w, a, \beta) \), where the weights \( w^* \) are obtained by minimizing the training loss \( \mathcal{L}_{\text{train}}(w, a, \beta) \). This implies a bilevel optimization problem with \( a, \beta \) as the upper-level variable and \( w \) as the lower-level variable. Evaluating the architecture gradient exactly can be prohibitive due to the expensive inner optimization. Therefore, the one-step approximation [19] is used and the approximate architecture gradient yields

\[
\nabla_{a, \beta} \mathcal{L}_{\text{val}}(w, a, \beta) = \xi \nabla^2_{a, \beta, w} \mathcal{L}_{\text{train}}(w, a, \beta) \nabla w \mathcal{L}_{\text{val}}(w, a, \beta)
\]

where \( w = w - \xi \nabla w \mathcal{L}_{\text{train}}(w, a, \beta) \) denotes the weights for one-step forward model. We use the first-order approximation following [32], w.r.t \( \xi = 0 \), to reduce the computational complexity in the search process.

#### 4.3.3 Batch Normalization

In Equation 5, the node \( j \) is the weighted sum of outputs in all operators in the operation space given \( i \) as the input. The contribution of a feature interaction \( o(\mathbf{X}(i)) \) can be measured by \( \text{softmax}(\sigma_o(i, j)) \). However, two problems may arise in fairness: (1) Our operation space contains both complicated and simple operators. In Bi-level optimization, the simple operators generally converge faster than complex operators, and then the stability of converged simple operators yield continuously better performance than unstable complex operators, which promotes larger weights \( \text{softmax}(\sigma_o(i, j)) \) on simple operators. Since each edge reserves only one operator with the largest weight, more simple operators are prone to be selected. Therefore, the optimization on architecture parameters \( a, \beta \) should consider the comparison fairness of converged and under-converged operators. (2) Since the range of \( o(\mathbf{X}(i)) \) varies among different operators and \( o(\mathbf{X}(i)) \) is jointly optimized with \( \text{softmax}(\sigma_o(i, j)) \), the coupling of their scale will lead to that \( \text{softmax}(\sigma_o(i, j)) \) can hardly represent the relative importance of \( o(\mathbf{X}(i)) \) [18].

It is well known that BN [11] can effectively solve the above problems by moving the output of each operator to a standard normal distribution, which is expressed as

\[
\text{BN}(\sigma(\mathbf{X})) = \sigma(\mathbf{X}) - \mu_{\mathcal{B}}(\sigma(\mathbf{X})) \frac{\sqrt{\sigma^2(\sigma(\mathbf{X})) + \epsilon}}{\sqrt{\sigma^2(\sigma(\mathbf{X}))}}
\]

where \( \mu_{\mathcal{B}} \) and \( \sigma_{\mathcal{B}} \) are the mean, standard deviation vector of \( \sigma(\mathbf{X}) \) on the field dimension in mini-batch \( \mathcal{B} \) and \( \epsilon \) is a constant for numerical stability. Note that there is no need to optimize to distribution after normalization. Therefore, we set the scale and shift parameters in BN to be 1 and 0, respectively.

### 4.4 Performance Evaluation

The objective of our method is to find powerful interactions that can achieve high predictive performance on a target data distribution. We obtain architecture parameters with the best performance on the validation set during the search process, and then convert the continuous architecture encoding to a discrete architecture. The discrete architecture will be trained from scratch. However, a well-known problem in weight-sharing NAS methods is the performance gap [31] between the continuous architecture encoding and the derived discrete architecture, which is called discrete loss. In this section, we first introduce the discretization technique, and then illustrate two optional training techniques for reducing the performance gap.
Where

We anneal the softmax

To comprehensively evaluate our method, we design plenty of experiments to answer the following research questions:

\[ \text{RQ1: How do the architectures identified via AutoPI perform compared with the SOTA hand-crafted models and available NAS algorithms on datasets with different scales?} \]

\[ \text{RQ2: How efficient is AutoPI compared with the SOTA hand-crafted models and available NAS algorithms?} \]

\[ \text{RQ3: How effective are training techniques in alleviating the performance gap between the networks before and after discretization?} \]

\[ \text{RQ4: How to understand the computational graph and operation space in our search space?} \]

### 4.4.1 Discretization

For each node in the converged computational graph, we retain the top-\(k\) strongest operators (from distinct nodes) among all candidate operators collected from all previous nodes. The strength of an operator is defined as

\[
\exp(\alpha_i) / \sum_{i' \in O} \exp(\alpha_{i'})
\]

We use \(k = 2\) for interaction cell and ensemble cell following existing works on CV domain [19, 32].

#### 4.4.2 Reducing Discrete Loss

Two training techniques can be optionally utilized to reduce the performance gap in our method.

1. **Temperature Anneal** [19] alleviate the performance gap via annealing the temperature parameter in softmax. By doing so, in each node pair, all operators have nearly the same strength defined above in order to train the weights \(w\) in the early search stage. As the temperature decreases, operators and edges begin to compete with each other to increase the discrepancy. We anneal the softmax temperature on every training epoch as follow formulation:

\[
\tau = \frac{1 + \log(T)}{1 + \log(t)}
\]

where \(T\) is the number of training epochs, and \(t \in [1, \ldots, T]\) is an index of the current epoch.

2. **Noisy Skip-connection** [4] proved that Skip-connection usually overperform other operators with softmax, resulting in a degenerate model where the skip connections contribute less. After injecting noise into the Skip-connection operator, the bias towards Skip-connection will be weakened. In this paper, we inject the Gaussian noise \(X \sim \mathcal{N}(\mu, \sigma)\) into our Skip-connection operator, i.e., \(\tilde{X} = X + \mu\) where \(\mu = 0\) and \(\sigma = \lambda \cdot \text{std}(X)\). Note that \(\lambda\) is a small positive coefficient.

## 5 Experiments and Results

To comprehensively evaluate our method, we design plenty of experiments to answer the following research questions:

\[ \text{RQ1: How do the architectures identified via AutoPI perform compared with the SOTA hand-crafted models and available NAS algorithms on datasets with different scales?} \]

\[ \text{RQ2: How efficient is AutoPI compared with the SOTA hand-crafted models and available NAS algorithms?} \]

\[ \text{RQ3: How effective are training techniques in alleviating the performance gap between the networks before and after discretization?} \]

\[ \text{RQ4: How to understand the computational graph and operation space in our search space?} \]

### 5.1 Experiment setup

**Datasets.** We conduct experiments on four public datasets with different statistical characteristics and five private commercial datasets generated from different E-commerce scenarios. The setting of public datasets follows previous works [3, 8, 14], including Criteo\(^2\), Avazu\(^3\), Movielens\(^4\) and Frappe\(^5\). We randomly split the instances by 8:1:1 for training, validation and testing, respectively. The commercial datasets contain users’ browsing and click records generated from real-world traffic logs on an E-commerce website. It is collected from five different scenes (S1-S5) in a discount activity. Each record has its corresponding user, item, context features, and a label indicating a click or not. Data of 3 days are used for training, and the following two days are used for validation and testing, respectively. The statistics of all datasets is shown in Table 1.

**Evaluation metrics.** We adopt two metrics for performance evaluation: AUC (Area Under the ROC curve) and Logloss (cross-entropy). Due to a large number of datasets, \(\Delta_{\text{AUC}}\) and \(\Delta_{\text{Logloss}}\) are also calculated to indicate averaged performance gain compared to a given benchmark over a group of datasets, which help to reflect the generalization ability of the algorithms. Note that an increase in AUC or decrease in Logloss at .001-level is known to be a significant improvement for the CTR prediction task [2, 3, 27].

**Comparison methods.** We compare AutoPI-D (AutoPI with Bi-level optimization in DARTS) with five classes of the existing approaches: (i) first-order approaches that is a weighted sum of raw features, including LR; (ii) FM-based methods that consider second-order cross features, including FM [25] and AFM [30]; (iii) advanced approaches that model higher-order feature interactions, including CrossNet [28], HOFM [1], NFM [8], PNN [22], CIN [14], AutoInt [27] and AFN [3]; (iv) ensemble models that are integrated with a tower of DNN, including Deep&Cross [28], Wide&Deep [2], DeepFM [7], xDeepFM [14], AutoInt+ [27] and AFN+ [3]; (v) NAS algorithms that automatically search the effective architecture for a target dataset, including AutoRec [29] and AutoPI-R (AutoPI with random search strategy). Note that no available codes have been found for other NAS methods, such as AutoCTR. We select the open-sourced AutoRec as our baseline. And we denote AutoRec-R, AutoRec-G, AutoRec-B as the random, greedy, Bayesian version of AutoRec, respectively.

**Implementation details.** We implement AutoPI using Pytorch\(^6\). Both interaction and ensemble cells consist of \(N = 4\) intermediate nodes. We set the embedding size as \(k = 10\) and adopt temperature anneal during the search process. Half of the training set is used for optimizing weights \(w\) in operators, and the other half is for architecture parameters \(\alpha, \beta\). The maximum number of training epochs for Criteo, Avazu, Commerical Dataset, Frappe, Movielens are 5, 5, 5, 50 and 100, respectively, and the corresponding batch size is 4096. We use momentum SGD to optimize the weights \(w\) and use Adam as the optimizer for architecture parameters \(\alpha, \beta\) with the configuration of [32]. To avoid overfitting, we perform early-stopping according to the Logloss on the validation set. Furthermore, we implement hand-crafted models according to the details of [3], where

---

\(^2\)http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

\(^3\)https://www.kaggle.com/c/avazu-ctr-prediction

\(^4\)https://grouplens.org/datasets/movielens/

\(^5\)http://baltrunas.info/research-menu/frappe

\(^6\)The code is available here: https://github.com/thu-media/AutoPI.
we use the same neural network structure (i.e., 3-layers MLP, 400-400-400) for all approaches that involve DNN. We implement the performance comparing with the AFN+, and outperforms all other different types of interactions is more effective than a single one. mentoring than individual models, which demonstrates that combining interactions are informative. Second, AFN and AutoInt consistently outperform FMs and HOFMs on all public datasets, which verifies interactions are informative. First, the methods that utilize higher-order feature interactions generally outperform those based on lower-order cross features, which means higher-order feature interactions are informative. Second, AFN and AutoInt consistently outperform FMs and HOFMs on all public datasets, which verifies that learning adaptive-order cross features can bring better predictive performance than modeling fixed-order feature interactions. Third, the ensemble models have a significant performance improvement than individual models, which demonstrates that combining different types of interactions is more effective than a single one. Moreover, the NAS baseline, AutoRec, produces the competitive performance than modeling fixed-order feature interactions. although the number of blocks and hyperparameters in the operator can be jointly optimized by different algorithms including the effective Bayesian method, the search space of AutoRec only incorporates limited efficient manual designs. AutoPI-D, exploring our search space by a simple search strategy, can search for more powerful interactions than AutoRec, which means our search space is more flexible and general than AutoRec. We further compare our method (AutoPI-D) with AutoPI-R. AutoPI-D yields a better or competing performance than AutoPI-R over all public datasets. This demonstrates that the improved gradient-based search strategy is more effective than the random method.

### Real Scenarios.
As illustrated in Table 1, the commercial datasets have close statistical properties but diverse semantics. We collect real-world traffic logs of five different scenes (S1-S5) from an online retailer platform. Each scene is a specific page on the E-commerce website to satisfy all kinds of shopping needs of users. The five scenes we collected contain food, makeup, etc. Each scene has different UI pages, target user groups, and data distribution. Due to page limitations, we select three representative hand-crafted models for real-scenario evaluation, including xDeepFM, AutoInt+ and AFN+.

### 5.2 Overall Performance (RQ1)
We set the powerful hand-crafted model, AFN+, as the benchmark for calculating $\Delta_{AUC}$ and $\Delta_{Logloss}$ on all datasets. Public Datasets. As illustrated in Table 1, the public datasets have distinct heterogeneous feature spaces, w.r.t. different number of instances, fields and features. Table 2 summarizes the performance of all methods on four public datasets.

We first present three important observations when comparing various hand-crafted models. First, the methods that utilize higher-order feature interactions generally outperform those based on lower-order cross features, which means higher-order feature interactions are informative. Second, AFN and AutoInt consistently outperform FMs and HOFMs on all public datasets, which verifies that learning adaptive-order cross features can bring better predictive performance than modeling fixed-order feature interactions. Third, the ensemble models have a significant performance improvement than individual models, which demonstrates that combining different types of interactions is more effective than a single one. Moreover, the NAS baseline, AutoRec, produces the competitive performance comparing with the AFN+, and outperforms all other hand-crafted models. This illustrates that using NAS can successfully find effective feature interactions and alleviate laborious and tedious architecture engineering. The search space in AutoRec supports the ensemble of at most 6 interactive blocks, where each block can be selected from one of the 5 types of configurable operators: MLP, FM, CrossNet, self-attention and element-wise interaction. Although the number of blocks and hyperparameters in the operators can be jointly optimized by different algorithms including the effective Bayesian method, the search space of AutoRec only incorporates limited efficient manual designs. AutoPI-D, exploring our search space by a simple search strategy, can search for more powerful interactions than AutoRec, which means our search space is more flexible and general than AutoRec. We further compare our method (AutoPI-D) with AutoPI-R. AutoPI-D yields a better or competing performance than AutoPI-R over all public datasets. This demonstrates that the improved gradient-based search strategy is more effective than the random method.

### Table 2: Effectiveness Comparison of Different Algorithms on Public Benchmarks.

| Model Class | Model | Criteo | Avazu | Movielens | Frappe |
|-------------|-------|--------|-------|-----------|--------|
|             |       | AUC    | Logloss | AUC | Logloss | AUC | Logloss |
| First-Order | LR    | 0.7858 | 0.4636 | 0.7313 | 0.4065 | 0.9215 | 0.3080 | 0.9329 | 0.2860 |
| Second-Order | FM    | 0.7933 | 0.4574 | 0.7496 | 0.3740 | 0.9388 | 0.2797 | 0.9641 | 0.2143 |
|             | AFM   | 0.7953 | 0.4554 | 0.7454 | 0.3766 | 0.9295 | 0.2836 | 0.9639 | 0.2294 |
| High-Order | CrossNet | 0.7915 | 0.4585 | 0.7498 | 0.3756 | 0.9323 | 0.2929 | 0.9393 | 0.2835 |
|             | HOFM  | 0.7960 | 0.4551 | 0.7516 | 0.3756 | 0.9410 | 0.3088 | 0.9709 | 0.2141 |
|             | NFM   | 0.7968 | 0.4537 | 0.7531 | 0.3761 | 0.9441 | 0.3004 | 0.9727 | 0.2079 |
|             | PNN   | 0.8026 | 0.4509 | 0.7526 | 0.3737 | 0.9469 | 0.2792 | 0.9735 | 0.2012 |
|             | CIN   | 0.8042 | 0.4472 | 0.7533 | 0.3756 | 0.9494 | 0.2600 | 0.9704 | 0.2342 |
|             | AutoInt | 0.8062 | 0.4456 | 0.7465 | 0.3790 | 0.9418 | 0.2762 | 0.9701 | 0.2034 |
|             | AFN   | 0.8061 | 0.4458 | 0.7512 | 0.3731 | 0.9477 | 0.2753 | 0.9759 | 0.1784 |
| Ensembled  | Deep&Cross | 0.8059 | 0.4463 | 0.7550 | 0.3721 | 0.9419 | 0.2791 | 0.9402 | 0.2808 |
|             | Wide&Deep | 0.8062 | 0.4453 | 0.7529 | 0.3744 | 0.9381 | 0.3310 | 0.9728 | 0.2038 |
|             | DeepFM  | 0.8025 | 0.4501 | 0.7535 | 0.3742 | 0.9424 | 0.3131 | 0.9719 | 0.2108 |
|             | xDeepFM | 0.8070 | 0.4443 | 0.7535 | 0.3737 | 0.9448 | 0.2717 | 0.9738 | 0.2098 |
|             | AutoInt+ | 0.8075 | 0.4438 | 0.7477 | 0.3776 | 0.9417 | 0.2764 | 0.9661 | 0.2102 |
|             | AFN+   | 0.8083 | 0.4437 | 0.7555 | 0.3718 | 0.9500 | 0.2585 | 0.9783 | 0.1762 |
| NAS        | AutoRec-R   | 0.8104 | 0.4412 | 0.7499 | 0.3746 | 0.9510 | 0.2573 | 0.9775 | 0.1688 |
|             | AutoRec-G   | 0.8100 | 0.4417 | 0.7467 | 0.3756 | 0.9496 | 0.2579 | 0.9771 | 0.1778 |
|             | AutoRec-B   | 0.8097 | 0.4418 | 0.7488 | 0.3766 | 0.9539 | 0.2443 | 0.9783 | 0.1610 |
|             | AutoPI-D   | 0.8105 | 0.4410 | 0.7598 | 0.3730 | 0.9555 | 0.2598 | 0.9820 | 0.1487 |
|             | AutoPI-R   | 0.8102 | 0.4420 | 0.7605 | 0.3691 | 0.9572 | 0.2660 | 0.9827 | 0.1426 |

$\Delta_{AUC}$ and $\Delta_{Logloss}$ for calculating $\Delta_{AUC}$ and $\Delta_{Logloss}$ on all datasets.
The comparison on the commercial datasets of these algorithms is shown in Table 3. From the results, we observe that the NAS methods outperform all hand-crafted models over all commercial datasets. This illustrates that the NAS methods are more effective than hand-crafted models on extensive real scenarios. Most importantly, our method increases the average AUC \( \Delta_{\text{AUC}} \) by 0.70%, which is larger than AutoRec by 0.50%, which further verifies the generalization of our proposed AutoPI.

### 5.3 Efficiency Analysis (RQ2)

In real-world scenarios, both the effectiveness and efficiency of a method are critical. Many SOTA methods can achieve good accuracy but fail to be deployed in commercial recommender systems due to their computational inefficiency, especially the additional time-consuming search process in NAS methods. In this section, we study the efficiency of AutoPI on public datasets using one NVIDIA P100 GPU. We present the search time of the NAS algorithms and the complexity (parameters size and Flops) of the identified best architectures in Table 4. Benefiting from the gradient-based search strategy and weight-sharing mechanism, our method (AutoPI-D) requires \( \sim 10 \) less search time than other NAS algorithms (AutoRec and AutoPI-D). In addition, the complexity of the architectures identified via our method are close to AutoRec, but much lower than hand-crafted models.\(^7\) Therefore, AutoPI can achieve better performance efficiently, which makes it easier to be deployed in industrial applications.

Furthermore, we present the ablation study of operators with large parameters in Table 5 to reduce the complexity of our architectures. In our operation space, MLP and FM have a large number of parameters, and we run four experiments on Avazu and Frappe with different configurations of operation space: (1) omitting MLP, (2) omitting FM, (3) omitting MLP and FM, (4) non-omitting. As illustrated in Table 5, the flexibility of AutoPI is validated. Therefore, by omitting operators with large parameters, the trade-off between performance and computational cost can be achieved.

### 5.4 Training Techniques Study (RQ3)

In this section, we evaluate the effectiveness of two training techniques on Avazu and Frappe by ablation study and the results are illustrated in Table 6. In detail, we run four experiments with different training techniques: (1) noisy skip-connection (Noisy): injecting the noise (\( \lambda = 0.03 \)) into the skip-connection with the fixed softmax temperature (\( \tau = 1 \)), (2) temperature anneal (Anneal): annealing the softmax temperature with the original skip-connection.

\(^7\)Although AutoInt+ has lower parameters than our method, the Flops of AutoInt+ is much larger than our method because of the key-value attention mechanism.
We run experiments of Avazu and Criteo multiple times with different random seeds. Since the obtained architectures identified by AutoPI are similar in each task, two representative architectures for Avazu and Criteo respectively are shown in Figure 4 for more detailed analysis of our search space. We analyse the architecture searched via AutoPI from three perspectives: (1) The order of interactions. The interaction cell in Avazu consists of a self-attention (Att) and seven skip-connections, building a second-order feature interaction, of which order is lower than Criteo. This demonstrates the Avazu relies more on lower-order cross features which is consistent with the observation in [3]. (2) The type of operators. In our operation space, there exist various operators with different complexity. We observe that compared to the architecture searched for Avazu, more complex operators are selected by AutoPI for Criteo, which means our method can automatically generate larger networks for more complicated datasets. (3) The integrated towers. An interesting observation is that the ensemble cell of Criteo has two repeated connections (i.e., 6, 7 → 8 and 6, 7 → 10), which can be simplified to a three-tower ensemble (i.e., integrating intermediate nodes 7, 8, 9). This illustrates that Criteo only needs a small number of towers to be integrated because of its higher-order interactions. On the contrary, Avazu needs more towers to be integrated because it prefers lower-order interactions.

### 6 Discussion

In this paper, we propose a general method AutoPI to automatically search for both effective and efficient architectures for CTR prediction in different scenarios with different scales. Extensive experiments on a diverse suite of benchmark datasets demonstrate the model searched by our method outperforms both the hand-crafted models and other NAS methods in terms of AUC and Logloss. Coupling with the gradient-based search strategy, AutoPI is more computationally efficient than all available NAS methods.

To reduce the performance gap between the continuous architecture encoding and discrete architecture, both temperature anneal and noisy skip-connection are verified to be effective. Finally, in the case study, some interesting observations regarding to the architecture properties for large-scale and small-scale datasets are summarized, which can provide insights for researchers on how to design better CTR models.

### Acknowledgments

This work was supported by NSFC under Grant No. 61936011 and No. 61521002, Beijing Key Lab of Networked Multimedia. Research reported in this publication was also supported by the National Library of Medicine of the National Institutes of Health, USA, under award number R01LM013151. We thank Tianchi Huang and Xin Yao for fruitful discussions. Finally, life is fine.

---

**Table 5: Ablation study of interactive operators.**

| Omitted Operation | Avazu | Frappe |
|-------------------|-------|--------|
|                   | AUC   | Logloss| AUC   | Logloss|
| MLP               | 0.7580| 0.3695 | 0.7580| 0.3708 |
| FM                | 0.7587| 0.3695 | 0.7587| 0.3708 |
| MLP+FM            | 0.7594| 0.3708 | 0.7605| 0.3715 |
| None              | 0.7605| 0.3715 | 0.7605| 0.3715 |

| Table 6: Effectiveness Comparison of Training Techniques. |
|-------------------|--------|-------|-------|-----|-----|
| Techniques        | Avazu  | Frappe| ΔAUC | ΔLogloss|
|                   | AUC    | Logloss|       |
| Origin            | 0.7606 | 0.3688| 0.9794| 0.1588|
| Noisy             | 0.7610 | 0.3682| 0.9809| 0.1522|
| Anneal            | 0.7605 | 0.3715| 0.9827| 0.1553|
| Noisy+Anneal      | 0.7605 | 0.3684| 0.9804| 0.1556|

**Figure 4: Two architectures found by AutoPI.**

---

**5.5 Case Study (RQ4)**

We run experiments of Avazu and Criteo multiple times with different random seeds. Since the obtained architectures identified by AutoPI are similar in each task, two representative architectures are generated.
References

[1] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. 2016. Higher-order factorization machines. arXiv preprint arXiv:1607.07195 (2016).

[2] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Harsh Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems. 7–10.

[3] Weiyu Cheng, Yanyan Shen, and Limeng Huang. 2020. Adaptive factorization network: Learning adaptive-order feature interactions. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 3609–3616.

[4] Xiangxiang Chu, Bo Zhang, and Xudong Li. 2020. Noisy differentiable architecture search. arXiv preprint arXiv:2005.03566 (2020).

[5] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. 2020. Fair darts: Eliminating unfair advantages in differentiable architecture search. In European Conference on Computer Vision. Springer, 465–480.

[6] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural architecture search: A survey. J. Mach. Learn. Res. 20, 55 (2019). 1–21.

[7] Huifung 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).

[8] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval. 555–564.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feed-forward networks are universal approximators. Neural networks 2, 5 (1989), 359–366.

[10] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. 2019. FiBiNET: combining feature importance and bilinear feature interaction for click-through rate prediction. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 3609–3616.

[11] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 448–456.

[12] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoRec: An Automated Recommender System. In Proceedings of the ADKDD’17. 1–7.

[13] Ting-Hsiang Wang, Xia Hu, Haifeng Jin, Qingquan Song, Xiaotian Han, and Zirui Liu. 2020. AutoRec: An Automated Recommender System. In Proceedings of the 33rd ACM SIGIR Conference on Research & Development in Information Retrieval. 199–208.

[14] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqing Chen, Wenyuanai, and Qiang Yang. 2019. Improved differentiable architecture selection in factorization models for click-through rate prediction. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2636–2645.

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

[16] Yanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqing Chen, Wenyuanai, and Qiang Yang. 2019. Improved differentiable architecture selection in factorization models for click-through rate prediction. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 199–208.