# AutoOpt: Automatic Hyperparameter Scheduling and Optimization for Deep Click-through Rate Prediction


Yujun Li*
Noah's Ark Lab
China
liyujun9@huawei.com

Xing Tang*
Noah's Ark Lab
China
xing.tang@huawei.com

Bo Chen*
Noah's Ark Lab
China
chenbo116@huawei.com

Yimin Huang
Noah's Ark Lab
China
yimin.huang@huawei.com

Ruiming Tang
Noah's Ark Lab
China
tangruiming@huawei.com

Zhenguo Li
Noah's Ark Lab
China
Li.Zhenguo@huawei.com



## ABSTRACT

Click-through Rate (CTR) prediction is essential for commercial recommender systems. Recently, to improve the prediction accuracy, plenty of deep learning-based CTR models have been proposed, which are sensitive to hyperparameters and difficult to optimize well. General hyperparameter optimization methods fix these hyperparameters across the entire model training and repeat them multiple times. This trial-and-error process not only leads to suboptimal performance but also requires non-trivial computation efforts. In this paper, we propose an automatic hyperparameters scheduling and optimization method for deep CTR models, *AutoOpt*, making the optimization process more stable and efficient. Specifically, the whole training regime is firstly divided into several consecutive stages, where a data-efficient model is learned to model the relation between model states and prediction performance. To optimize the stage-wise hyperparameters, AutoOpt uses the *global* and *local* scheduling modules to propose proper hyperparameters for the next stage based on the training in the current stage. Extensive experiments on three public benchmarks are conducted to validate the effectiveness of AutoOpt. Moreover, AutoOpt has been deployed onto an advertising platform and a music platform, where online A/B tests also demonstrate superior improvement. In addition, the code of our algorithm is publicly available in MindSpore[1].



## CCS CONCEPTS

• **Information systems** → **Display advertising**; *Recommender systems.*

## KEYWORDS

CTR Prediction, Hyperparameter optimization, Recommendation, Online advertising

**ACM Reference Format:**
Yujun Li, Xing Tang, Bo Chen, Yimin Huang, Ruiming Tang, and Zhenguo Li. 2023. AutoOpt: Automatic Hyperparameter Scheduling and Optimization for Deep Click-through Rate Prediction. In *Seventeenth ACM Conference on Recommender Systems (RecSys '23), September 18–22, 2023, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3604915.3608800


## 1 INTRODUCTION

Click-through rate (CTR) prediction is critically important for recommender systems and online advertisement systems [3, 14, 34]. Recently, plenty of deep learning-based CTR models are proposed to improve the prediction accuracy, including FNN [43], DeepFM [13], DCN [40], and so on. Generally, these deep CTR models mainly consist of three main components: **embedding component** that projects input features into latent space [29]; **feature interaction component** that captures explicit feature interactive signals [2, 28]; and **deep component** that models sophisticated high-order implicit feature interactions [8, 13].

However, training a deep CTR model well and giving full play to its advantages is not easy [42]. *Firstly*, during the model training procedure, setting appropriate optimizer parameters (e.g., learning rate) contributes to the efficient learning of model parameters, which is highly related to the convergence speed and prediction effect. *Secondly*, due to the data sparsity problem that widely exists in CTR prediction, regularization technique (e.g., $L_1$ and $L_2$ normalization [22], dropout [15]) is introduced to improve the generalization of deep neural networks [30]. Therefore, it is of great significance to provide proper regularization strength for alleviating this issue. *Thirdly*, for deep CTR models, different components face diverse degrees of sparsity. For example, embedding components suffer from sparse input features [35] (like long-tailed features and cold-start users), while feature interaction and deep components may encounter more severe sparsity problems due to the less-frequent high-order interactions [26]. Hence, it is vitally important to perform component-level regularization in a fine-grained manner, so that effective training procedures for deep CTR models can be obtained. Consequently, searching appropriate optimization-related hyperparameters is conductive to improving the performance of

*All authors contributed equally to this research.
[1]https://gitee.com/mindspore/models/tree/master/research/recommend/autoopt



*RecSys '23, September 18–22, 2023, Singapore, Singapore*



ACM ISBN 979-8-4007-0241-9/23/09…$15.00
https://doi.org/10.1145/3604915.3608800

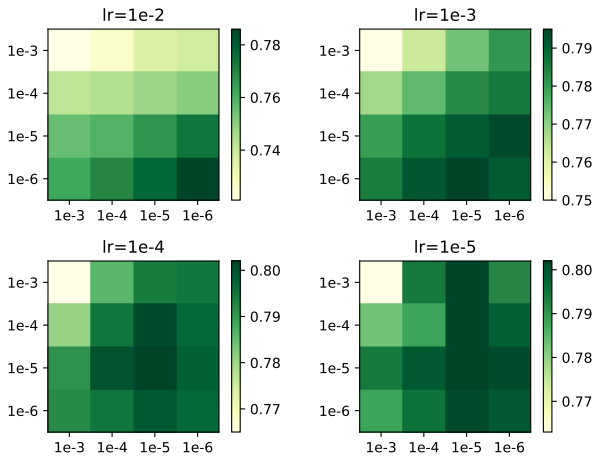


**Figure 1: An example of grid search on regularization of IPNN. The X-axis represents the coefficient of embedding $\lambda_{emb}$, The Y-axis is the coefficient of deep $\lambda_{deep}$.**

deep CTR models. Figure 1 indicates that the prediction accuracy of deep CTR models varies significantly with the learning rate of optimizer and component-level regularization coefficients.

Searching optimal *structure-related hyperparameters* for deep CTR models has made great progress, such as embedding dimension search [19, 46], neural architecture search, including feature interactions [47], network layers and width [45], activation functions [45], as well as operation functions [37]. Nevertheless, prior work on how to search suitable *optimization-related hyperparameters* (including learning rate, component-level $L_1$, $L_2$ normalization coefficients, dropout ratio, etc.) is scarce, motivating us to conduct in-depth research. A straightforward method is grid search [16], which shrinks to a discrete search space with $n$ candidate values for finding the optimal parameters brutely. However, when a deep CTR model contains $m$ optimization-related hyperparameters that need to be searched, the complexity of search space by grid search is $O(n^m)$, which requires a huge amount of computation to find the optimal hyperparameters. Moreover, limited by the grid size, the optimal hyperparameters existing in the continuous search space are difficult to be explored. As illustrated in Figure 1, it is highly possible that even better performance can be obtained outside the search space, such as optimal $\lambda_{deep}$ lying between $10^{-5}$ and $10^{-6}$. To overcome the disadvantages of the grid search, some general hyperparameter optimization (*HPO*) methods are proposed [10]. Generally, the whole model training process is repeated multiple times to search for the optimal hyperparameters. Combining Bayesian optimization with the hyperband strategy, BOHB [9] reduces the repeated times of training. Nevertheless, repeating the whole training process multiple times is tedious and costs too much computation, because CTR prediction is a time-sensitive task and requires fast training to meet periodic model update requirements. Besides, existing search methods fix hyperparameters in one single trial-and-error process, which lacks exploiting valuable information during training. Some research has empirically demonstrated that hyperparameter scheduling helps to improve the model performance [5, 38].

In this paper, to tackle the above issues raised by previous studies, we propose **AutoOpt**, an **Auto**matic hyperparameters scheduling and **Opt**imization method for training deep CTR models. To exploit the dynamic training information, we reformulate the hyperparameter optimization problem as several consecutive subproblems. Specifically, AutoOpt divides the entire training process into several stages, where the optimization-related hyperparameters are held constant in each stage and the proposed hyperparameters for the next stage are learned based on the previous one. To adjust the stage-wise hyperparameters more efficiently and stably, AutoOpt utilizes multi-worker parallel training to provide more modeling data. Specifically, each stage consists of three phases: *model training* in the current stage, *global scheduling* based on parallel training dynamic, and *local scheduling* based on the current best result. In all, the contributions of our work are summarized as follows:

- We propose AutoOpt to automatically search suitable optimization-related hyperparameters for deep CTR prediction. The component-level regularization is performed in a fine-grained manner for achieving an effective and stable model training procedure.
- We provide a stage-wise approach to search and schedule the hyperparameters. Moreover, a multi-worker parallel training system is further proposed to make the search process more stable and efficient.
- Extensive experiments offline and online validate the effectiveness of AutoOpt. Moreover, AutoOpt has been deployed on a real-world system of millions of users and improves the performance of several mainstream deep CTR models.

We organize the rest of this paper as follows. In Section 2, we briefly review related works. In Section 3, we give some preliminaries on CTR models. In Section 4, we present our method AutoOpt. Section 5 details the experimental setting and corresponding results on benchmarks. Online results deploying AutoOpt are demonstrated in Section 6. Finally, we conclude this work in Section 7.

## 2 RELATED WORK

### 2.1 Deep CTR Prediction

Deep CTR prediction has been investigated a lot in recent years [44]. Factorization-Machine Supported Neural Networks (FNN) [43] is a forward neural network using factorization machine (FM) [33] as the pretrained embedding layer. However, this network only takes the high-order implicit feature interaction into consideration. Wide&Deep (WDL) [8] was initially proposed by Google, which jointly learns linear models and deep neural networks to model feature interactions. In this work, feature engineering is required, which depends on expertise experience. To overcome this, DeepFM [13] replaces the wide part of WDL with FM and shares the feature embedding between the FM and deep component. To improve the performance, many works focus on modeling feature interaction explicitly. DCN [40] efficiently captures feature interactions with cross layers. Similarly, xDeepFM [25] proposes a novel Compressed Interaction Network (CIN) to model both the low-order and high-order feature interactions in an explicit way. However, training a deep CTR model well is not easy because several optimization-related parameters need to be carefully tuned. To achieve this, we propose AutoOpt to automatically search suitable optimization-related hyperparameters for deep CTR models.

## 2.2 Hyperparameter Optimization

For deep models, the involved hyperparameters can be divided into two groups: structure-related hyperparameters (e.g., embedding size, layers of networks) and optimization-related hyperparameters (e.g., learning rate, regularization coefficient). Searching optimal structure-related hyperparameters automatically has made great progress, such as embedding dimension search in NIS [19] and AutoDim [46], neural architecture search, including feature interactions in AIM [47], network layers and width in AMEIR [45], activation functions in AMEIR [45], as well as operation functions in AutoCTR [37]. Nevertheless, researchers seldom set foot in the context of searching suitable optimization-related hyperparameters for CTR models. To avoid the huge computation of grid search, Auto$L_2$ [24] proposes a dynamical schedule for the regularization parameter $L_2$ automatically. Bayesian optimization (BO) [4] is a widely studied approach that adjusts the hyperparameters according to the model evaluation. However, it is not time-efficient since each evaluation requires a whole training process. BOHB [9] combines Bayesian optimization with successive halving which pays more attention to potential hyperparameters. Besides, PBT [17] improves time-efficiency by parallelly exploring hyperparameter space. But it consumes a lot of computing resources since it requires 32 or more workers to conduct sufficient exploration. AutoLRS [18] reduces the computation time of model evaluation by mutual-training between BO and loss forecasting model. However, it will suffer unstable training due to the uncertainty of the loss forecasting model. $\lambda$opt [7] is the first to introduce fine-grained regularization into matrix factorization for recommender systems. However, it cannot be applied to deep CTR models due to its complex computation. These existing methods keep the hyperparameters of regularization fixed during training, while our method adjusts the hyperparameters adaptively along with the training, which is helpful to improve the model performance [5].

# 3 PRELIMINARY

For the CTR prediction task, the data is collected as multi-field, where each feature field contains either categorical values or numerical values. In the pre-processing phase, numerical features are usually transformed into categorical form by bucketing or some other methods [12]. Hence, the input features are denoted as $\mathbf{x} = [x_1, x_2, ..., x_m]$, where $m$ is the number of fields. Then, an **embedding layer** is applied upon the multi-field input data, transforming each raw feature to a dense real-value vector as,

$$\boldsymbol{e}_i = \boldsymbol{V}_i \mathbf{x}_i,$$

where $\boldsymbol{V}_i \in \mathbb{R}^{n_i \times d}$ is the embedding table for the $i$-th field, $\mathbf{x}_i$ is the one-hot vector, $n_i$ is the total number of feature values and $d$ is the embedding dimension. Then the $m$ vectors are concatenated into $\boldsymbol{e} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m]$.

To improve model performance, deep CTR models usually employ various **feature interaction layers** to explicitly learn the feature interactions. An example is the cross layer in DCN[40], which is denoted as follows:

$$\boldsymbol{x}_k = \boldsymbol{x}_0 \boldsymbol{x}_{k-1}^T \boldsymbol{w}_k + \boldsymbol{b}_k + \boldsymbol{x}_{k-1},$$

where $\boldsymbol{x}_k \in \mathbb{R}^d$ is the vector denoting the output from the $k$-th cross layer, and $\boldsymbol{w}_k, \boldsymbol{b}_k \in \mathbb{R}^d$ are the weight and bias parameters of the $k$-th layer.

Besides, another important component **deep layer** is utilized to model high-order feature interactions implicitly, which is formulated as:

$$\boldsymbol{x}_k = \sigma\left(\boldsymbol{W}^k \boldsymbol{x}_{k-1} + \boldsymbol{b}^k\right),$$

where $\boldsymbol{W}^k$ and $\boldsymbol{b}^k$ are the weight and bias of the $k$-th deep layer respectively, $\sigma$ is an activation function. Finally, an output layer is used to predict the click probability $\hat{y}$ for each instance [23]. A widely-used *Logloss* is adopted for training the model, which is noted as:

$$Logloss = -\frac{1}{Q}\sum_{i=1}^{Q}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)),$$

where $y_i$ is the ground truth of user click label and $\hat{y}_i$ is the estimated value of the $i$-th instance, respectively. $Q$ is the total number of training instances in the training set. With input data $(\boldsymbol{x}, y)$ following the training data distribution, we define the loss function with regularization to ensure the generality,

$$L_{train}(\Theta|\lambda) = \mathbb{E}_{(\boldsymbol{x},y)} LogLoss(\boldsymbol{x}, y \mid \Theta) + \Omega(\Theta \mid \lambda),$$

where $\Theta$ is model parameters consisting of embedding component $\boldsymbol{V}$, feature interaction component $\boldsymbol{w}$ and deep component $\boldsymbol{W}$. $\Omega(\cdot)$ denotes regularization terms. $\lambda$ denotes different coefficients for component-level regularization.

# 4 THE PROPOSED METHODOLOGY

In this section, we first formulate the hyperparameter scheduling problem. Based on the formulation, we describe our proposed AutoOpt framework.

## 4.1 Problem Formulation

As mentioned in Section 3, training a deep CTR model well is not easy, which requires the correct tuning of many parameters, such as regularization coefficients $\lambda$, learning rate $\alpha$ and dropout ratio $\varepsilon$ [24]. To search the optimal optimization-related parameters $\Lambda = (\lambda, \alpha, \varepsilon)$, the computation procedure is stated as the following optimization problem,

$$\min_{\Lambda} L_{val}(\Theta(\Lambda)) \tag{1}$$

$$\text{s.t.} \quad \Theta(\Lambda) = \arg\min L_{train}(\Theta \mid \Lambda). \tag{2}$$

Eq.(1) is the outer optimization where $L_{val}(\cdot)$ indicates the model performance on validation data. Eq.(2) is the inner optimization with optimization-related parameters $\Lambda$ which optimizes the model parameters $\Theta$.

The above optimization problem shows that the goal of hyperparameter tuning is to find the optimal $\Lambda$ to make the model obtain the optimal performance on the validation dataset. A naive method is to optimize the model in Eq.(2) over the whole training data with every possible hyperparameter via grid search and then choose the one with the best performance evaluated on Eq.(1). However, the search overhead is unaffordable in practice due to time and resource constraints in practical CTR prediction. Besides, changing the suitable hyperparameters during the model training procedure

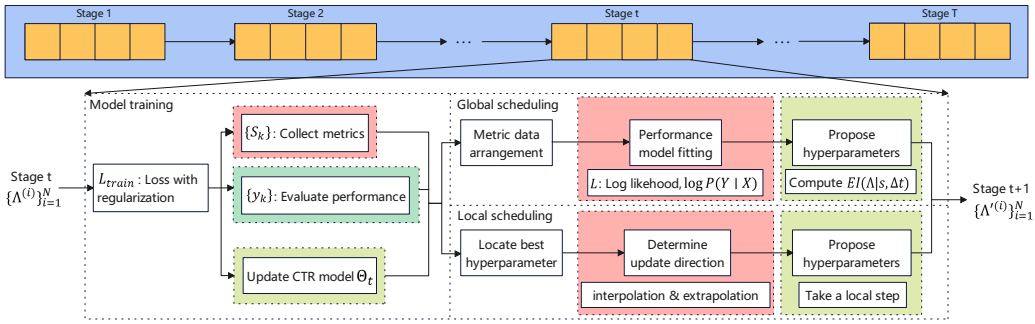


**Figure 2: One stage of AutoOpt. Model training phase trains CTR models with hyperparameters from the previous stage. Performance fitting phase fits a model of performance. Hyperparameter proposal phase proposes potential hyperparameters for the next stage.**

is conducive to accelerating convergence and improving CTR prediction performance.

To achieve both time-efficient and resource-efficient hyperparameter search for CTR models, we split one full training run into $T$ stages (shown in the blue block in Figure 2), where each stage has $K$ epochs. At the $t$-th stage, the model parameters $\Theta$ and optimization-related hyperparameters $\Lambda$ are initialized with $\Theta_t$ and $\Lambda_t$ respectively, and the optimization objective can be formulated as follows,

$$\min_{\Theta_t} \; L_{train}(\Theta_t|\Lambda_t) \tag{3}$$

$$s.t.\ \Lambda_t = \Phi(\Lambda_{t-1}, \Theta_t) \tag{4}$$

$$\Theta = \{\boldsymbol{V}, \boldsymbol{w}, \boldsymbol{W}\} \tag{5}$$

$$\Lambda = \{\lambda_{emb}, \lambda_{deep}, \lambda_{inter}, \alpha, \varepsilon, \dots\}, \tag{6}$$

where $\Theta_t$ is updated by $L_{train}$ for $K$ epochs, $\Phi$ is the scheduler to update $\Lambda_t$ based on the previous $\Lambda_{t-1}$. At the next stage, the model is initialized with $(\Theta_{t+1}, \Lambda_{t+1})$ and repeats the optimization procedure. By dividing the whole optimization procedure into multiple optimization stages, the search process can be significantly accelerated. Moreover, the hyperparameters can be optimized dynamically according to the model rewards received from different stages.

## 4.2 AutoOpt

In this section, we propose an automatic hyperparameters scheduling and optimization method to solve the above problem. We start with a quick introduction to the system design and then go through the details of each part.

*4.2.1 System Design.* To improve the stability and efficiency of hyperparameter exploration in AutoOpt, we propose a **multi-worker parallel training** scheme (shown in Figure 2), and the algorithm is presented in Algorithm 1. The entire training process is divided into multiple stages, where AutoOpt deploys $N$ workers for parallel training in each stage. Each stage consists of three phases. Phase 1 is the model training, each worker initializes the model weight with the best model checkpoint from the last stage and updates model weights $\Theta$ with an assigned hyperparameter $\Lambda$. In Phase 2, the global scheduling module accumulates training metrics collected from $N$ workers and learns a performance prediction model to propose potential hyperparameters. And in Phase 3, the local scheduling module updates the current best hyperparameter in a heuristic way.

**Algorithm 1** The AutoOpt Algorithm

1: **Input:** Hyperparameter space $\mathcal{H}$, worker number $N$, stage number $T$, epoch number $K$ for each stage, initial network weight $\Theta_0$.
2: Initialize hyperparameters by sampling $N$ hyperparameters from $\mathcal{H}$ and setting the $i$-th worker $\Lambda^{(i)}$. Initialize network weights of each worker as $\Theta_0^{(i)} = \Theta_0$.
3: **for** $t = 1, ..., T$ **do**
4:  **for** $i = 1, ..., N$ **do**
5:  With $\Lambda^{(i)}$, update the $i$-th worker's network weights in *Model Training*.
6:  Collect metrics $\mathcal{S}^{(i)}$ along the training process.
7:  **end for**
8:  The global scheduling module gathers metrics of $N$ workers, learns a performance prediction model, and proposes $N-1$ potential hyperparameters from the entire search space.
9:  The local scheduling module locates the current best hyperparameter $\Lambda^*$, determines the update direction in two cases, and proposes one hyperparameter by taking a local step.
10:  Return $N$ new hyperparameters $\{\Lambda'^{(i)}\}_{i=1}^N$ for the next stage.
11: **end for**

*4.2.2 Model Training.* At the first stage, $N$ workers are initialized with randomized hyperparameters. At the following stages, $N$ workers are configured with hyperparameters from the global and local scheduling modules. To make the CTR model have a good prediction ability to facilitate subsequent hyperparameter search, the first phase in each stage is to train the CTR model in $K$ epochs over the training data and update model weights. Then, an evaluation process is performed to estimate the performance of the warmed-up model, and several metrics are collected to present the prediction performance. Specifically, for the $k$-th epoch, we use the metrics including *training loss, training AUC, validation loss* to represent the model state, i.e., $\boldsymbol{s}_k = \{loss_k^{train}, auc_k^{train}, loss_k^{val}\}$, and use the *validation AUC* as indicator to reflect its prediction performance, i.e., $y_k = auc_k^{val}$. Note that the model weights $\Theta$ are excluded due to the limited extra computation budget.

*4.2.3 Global Scheduling.* The global scheduling process aims to propose potential hyperparameters from the entire search space in a global optimal perspective. To achieve this, we leverage the metrics collected from the model training phase that reflect the model states $s_k$ and performance $y_k$ of the current stage, to build a performance model. Specifically, the collected metrics along the model training phase can be regarded as the time series of model performance with respect to states. Therefore, given a fixed hyperparameter $\Lambda$, to capture the relation between model state $s_i$ and the subsequent model performance $y_{i+\Delta t}$ of the CTR model after $\Delta t$ epochs, a performance prediction model $f(\cdot)$ is proposed:

$$f(\Lambda, s_i, \Delta t) = y_{i+\Delta t}. \tag{7}$$

To learn an effective regression model $f(\cdot)$, two issues should be carefully considered. Firstly, data used to train the prediction model is collected along the model training, making only a small number of samples available. As a result, many commonly used regression models [6] which rely on a large amount of data cannot satisfy this requirement. Secondly, the collected data is noisy due to the unstable model training process. To overcome these issues, AutoOpt leverages Gaussian process [11] to capture the relation between model states and model performance, which assumes that models with close hyperparameters have nearly similar performance [9, 10].

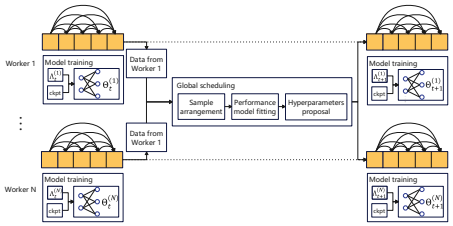


**Figure 3: A window sliding strategy to arrange samples for performance prediction model.**

*Sample Arrangement.* The samples for the performance prediction model $f(\cdot)$ are accumulated by a window sliding strategy at each stage, as shown in Figure 3. For each worker after $K$ epochs model training in the $t$-th stage, we can collect $K$ model states $\{s_1, s_2, ..., s_K\}$ and $K$ performance results $\{y_1, y_2, ..., y_K\}$. Then, the samples of the $t$-th stage are constructed as $\{\{(\Lambda_t, s_0, \Delta t, y_{0+\Delta t})\}_{\Delta t=1}^{K}, \{(\Lambda_t, s_1, \Delta t, y_{1+\Delta t})\}_{\Delta t=1}^{K-1}, \ldots, \{(\Lambda_t, s_{K-1}, \Delta t, y_{K-1+\Delta t})\}_{\Delta t=1}^{1}\}$, where $s_0$ is the model state of the last stage, $\Delta t$ denotes the epoch number. As a result, $N$ workers collect $N(K+1)K/2$ samples in each stage. We renumber the samples with $M = N(K+1)K/2$, and thus get training samples $\mathcal{S} = \{(\Lambda_1, s_1, \Delta t_1, y_1), (\Lambda_2, s_2, \Delta t_2, y_2), ..., (\Lambda_M, s_M, \Delta t_M, y_M)\}$. Thanks to the multi-worker parallel training framework, more training data can be collected, facilitating the fitting of the performance prediction model.

Specifically, assume that the samples follow a multivariate Gaussian distribution with a mean of zero,

$$[y_1, y_2, ..., y_M] \sim \mathcal{N}(0, \Sigma), \tag{8}$$

where covariance matrix $\Sigma$ has $\Sigma_{ij} = \mathcal{K}([\Lambda_i, s_i, \Delta t_i], [\Lambda_j, s_j, \Delta t_j])$ and $\mathcal{K}(\cdot, \cdot)$ is the kernel function. We choose an automatic relevance determination form of the kernel [32] to express each dimension as being independent of others,

$$\mathcal{K}(x, x') = \sigma_y^2 \exp\Big(\sum_{p=1}^{d_1+d_2+1} -\frac{(x_{[p]} - x'_{[p]})^2}{2l_p^2}\Big), \tag{9}$$

where $x = (\Lambda, s, \Delta_t)$ denotes a vector concatenated by $\Lambda \in \mathbb{R}^{d_1}$, $s \in \mathbb{R}^{d_2}$ and $\Delta t \in \mathbb{R}$, $x_{[p]}$ is the $p$-th feature in $x$. $\sigma_y$ is the signal variance and $l_p$ is the length scale. For convenience, let $X = [x_1, x_2, \ldots, x_M]^\top \in \mathbb{R}^{M\times(d_1+d_2+1)}$ and $Y = [y_1, y_2, ..., y_M]^\top \in \mathbb{R}^M$. To learn the prediction model, we can maximize the log-likelihood,

$$\max_{\sigma_y, l_p} \log P(Y|X) = -\frac{1}{2}Y^\top \Sigma^{-1} Y - \frac{1}{2}\log|\Sigma| - \frac{M}{2}\log 2\pi. \tag{10}$$

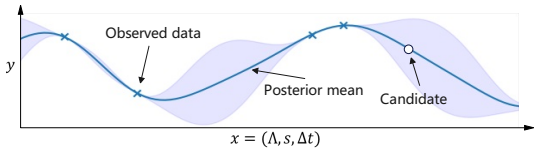


**Figure 4: Gaussian process for performance prediction. The curve is the posterior mean of different candidate hyperparameters. The shaded area shows the 95% confidence interval.**

*Hyperparameter Proposal.* After the performance fitting phase, AutoOpt is dedicated to proposing hyperparameters for the next stage. As illustrated in Figure 4, we can get the posterior distribution of the performance prediction,

$$P(y \mid X, Y, \Lambda, s, \Delta t) = \mathcal{N}(k_1^\top \Sigma^{-1} Y, k_{11} - k_1^\top \Sigma^{-1} k_1), \tag{11}$$

where $k_1 = [\mathcal{K}(x, x_1), ..., \mathcal{K}(x, x_M)]^\top$ denotes the vector of covariances between the current state and the $M$ training states, and $k_{11} = \mathcal{K}(x, x)$ denotes the covariance of the current state. Given model state $s$ and epoch number $\Delta t$, every possible hyperparameter $\Lambda$ corresponds to a different posterior distribution of $y$. The goal is to obtain an expected advantage of the predicted performance over the baseline as high as possible. Therefore, AutoOpt chooses expected improvement to select scheduling hyperparameters,

$$EI(\Lambda \mid s, \Delta t) = \int [y - y^*]_+ P(y|X, Y, \Lambda, s, \Delta t)\mathrm{d}y, \tag{12}$$

where the state with the best performance is chosen as anchor state $s$, $[\cdot]_+ = \max(\cdot, 0)$, $y^*$ is the reference performance which is set to be the current best performance.

*4.2.4 Local Scheduling.* The global scheduling process proposes the potential hyperparameters globally from the entire search space by fitting a performance model. The effectiveness of the proposed hyperparameters depends on the accuracy of the performance model. Although the multi-worker parallel training scheme provides more training data to improve prediction accuracy and ensures adequate hyperparameter exploration, resource efficiency is also important for industrial applications. To achieve the balance between efficiency and effectiveness, AutoOpt deploys a heuristically local scheduling algorithm to search the hyperparameters and guarantee ample exploitation. Suppose that the underlying relationship between model performance and hyperparameters is smooth.

Therefore, AutoOpt aims to update the current best hyperparameter towards a good direction at the next stage. The direction is decided by a heuristically local scheduling algorithm with interpolation.

The local scheduling algorithm takes a coordinate way to update each coordinate of hyperparameters. Firstly, AutoOpt finds the current best hyperparameters $\Lambda^*$ with the highest validation AUC,

$$\Lambda^* = \arg\max_{\Lambda \in \Lambda_C} \max_{\Delta t} \text{ValidAUC}(\Lambda, \Delta t), \tag{13}$$

where $\Lambda_C$ denotes all the hyperparameters of $N$ workers. Then, the update direction $d$ should be determined. Finally, a local step is taken to update $\Lambda^*$ towards the heuristic direction $d$.

As shown in Figure 5, for the $i$-th coordinate, there are two cases according to the location of $\Lambda_i^*$: (1) $\Lambda_i^*$ is not the closest to the boundaries of search space (left subfigure); (2) $\Lambda_i^*$ is the closest to the boundaries of search space (right subfigure). The two nearest points to $\Lambda_i^*$ are $\Lambda_i'$ and $\Lambda_i''$.

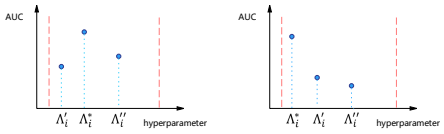


**Figure 5: Examples for Case (1) and (2). The red lines represent search space boundaries of the $i$-th coordinate.**

In Case (1), we choose the Lagrange interpolation method to get a polynomial through the tree points and obtain a quadratic curve. Note that the optimal point $\widehat{\Lambda}_i^*$ along the curve locates between $\Lambda_i'$ and $\Lambda_i''$ since the quadratic curve is unimodal. The update direction is given by $d = sgn(\widehat{\Lambda}_i^* - \Lambda_i^*)$, where $sgn(\cdot)$ is a sign function.

As for infrequent Case (2), $\Lambda_i^*$ is the closest to the boundary of the search space. Therefore, the coordinate search space should be shifted or set larger. Thus, we take an optimistic update by pushing the next hyperparameter closer to the boundary and choose a simple linear extrapolation to decide its update direction by $d = sgn(\Lambda_i^* - \Lambda_i')$. Note that $sgn(\Lambda_i^* - \Lambda_i') = sgn(\Lambda_i^* - \Lambda_i'')$.

# 5 EXPERIMENTS

## 5.1 Experiment Setup

*5.1.1 Dataset.* We evaluate the performance of our method on three benchmark datasets in this paper. The basic statistics are presented in Table 1.

**Criteo**[2]: It is one week of display advertising data released by CriteoLab, which is widely used in CTR prediction. The data contains nearly 45 million click records, which contain 13 numerical feature fields and 26 categorical feature fields. We set the categories appearing less than 100 times as a dummy category "other".

**Avazu**[3]: This dataset was provided by Avazu to predict whether a mobile ad will be clicked. It contains about 40 million click logs in 10 days with 23 categorical feature fields. Categorical features with less than 20 times of appearance are replaced by the dummy category "other".

[2]http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/.
[3]http://www.kaggle.com/c/avazu-ctr-prediction.

**KDD12**[4]: It is published in KDD-cup 2012 track 2 and investigated in FFM [20]. To reduce the dimensionality, we sample the data with features appearing more than 15 times. The sampled data contains 11 categorical fields and more than 37 million click logs.

**Table 1: Statistics of the datasets**

| Dataset | #Instances | #Categories | #Fields | Pos Ratio |
|---|---|---|---|---|
| Criteo | 45,840,000 | 184,964 | 39 | 0.26 |
| Avazu | 40,428,000 | 211,610 | 23 | 0.17 |
| KDD12 | 37,358,042 | 3,747,904 | 11 | 0.05 |

Note that we split all these datasets randomly into three parts: 80% for training, while 10% for validation and 10% for testing, following [25].

*5.1.2 Evaluation Metrics.* The evaluation metrics are **AUC** (Area Under ROC) and **log-loss** (cross-entropy). AUC is a widely used metric for evaluating CTR prediction, and a larger value indicates a better result. Log-loss is the loss in CTR prediction, which is also a widely used metric in binary classification, and a smaller value means better performance. Note that an improvement of 0.1% in AUC is usually regarded as significant for the CTR prediction [39]. All the experiments are repeated 5 times to get the average performance. The two-tailed unpaired $t$-test is performed to detect a significant difference between AutoOpt and the best baseline.

*5.1.3 Baseline Methods and Implementation Details.* We compare AutoOpt and seven baseline methods, including LR decay [41], two variants of grid search, PBT [17], AutoLRS [18], BO [36], BOHB [9], AutoL2 [24] on several mainstream deep CTR models. There are brief introductions to these methods.

- LR-Decay is a popular technique for training neural networks, which decays the learning rate by a certain factor after predefined epochs.
- Grid-Same is the grid search method, where a global regularization coefficient is used for different components.
- Grid-Componet is the grid search method, where component-level regularization coefficients are searched for different components.
- PBT uses multiple workers to explore hyperparameter space and update hyperparameters by adding noises [17].
- AutoLRS reduces the computation time of model evaluation by mutual-training between BO and loss forecasting model.
- BO is a sequential model-based approach to find the global optimal value [36], which can be used to tune the hyperparameters in recommender systems [10].
- BOHB combines the benefits of both BO and Hyperband [9], which is commonly used for hyperparameters search due to its efficiency and effectiveness.
- AutoL2 is a recently proposed method for adaptively adjusting the $L_2$ regularization term, which has achieved good results in computer vision [24].

All of the parameters in these methods are fine tuned in our experiments. To fully express the compatibility, we apply various

[4]http://www.kddcup2012.org/c/kddcup2012-track2/data

mainstream deep CTR models in recent research as backbone models: DeepFM [13], IPNN [31], DCN [40], and xDeepFM [25]. In IPNN, regularization for embedding components and deep components is required. For DeepFM, DCN, and xDeepFM, a regularization for feature interaction component is introduced besides the embedding component and deep component.

*5.1.4 Search Space.* Following the search space in [25] and [31], we define the continuous search space of $\Lambda$ as listed below,

- $\lambda_{emb} \in [1e-7, 1e-3]$,
- $\lambda_{deep} \in [1e-7, 1e-3]$,
- $\lambda_{inter} \in [1e-7, 1e-3]$,
- $\alpha \in [1e-6, 1e-2]$,
- $\varepsilon \in [0.5, 1]$.

The search space is also used for AutoLRS, BO, BOHB, and AutoL2. As for the grid search, we set the search step as 0.5 in the logarithmic scale, which produces a discrete search space.

*5.1.5 Implementation Details.* For a fair comparison, the embedding dimension is set to 64 for Criteo and Avazu dataset, and 16 for the KDD12 dataset. For the optimization method, we use the Adam [21] with a mini-batch size of 1000 for all the datasets. The hidden layers of the deep component are fix to 512-256-128 by default, and all activation functions are RELU. Besides, the explicit feature interactions modeling in DCN and xDeepFM (namely, cross layer and CIN) are set to 2 layers. For AutoOpt, the number of stages is set to $T = 10$, the epoch number of each stage is set to $K = 5$, and the number of workers is set to $N = 8$ by default. We conduct our experiments on a Linux server with 18 Intel Xeon Gold 6154 cores, 128 GB memory, and four NVIDIA-V100 GPUs with PCIe connections.

## 5.2 Overall Performance

Table 2 presents the overall performance of AutoOpt and other baselines with several mainstream backbone models on three datasets. The observations can be summarized as follows.

- AutoOpt outperforms all the SOTA baselines over three datasets by a significant margin. Besides, the improved performance of AutoOpt over various backbone CTR models is significant and consistent, demonstrating the superiority and robustness in searching the optimal hyperparameters adaptively.
- Grid-Component performs better than Grid-Same, which indicates that component-level regularization is important for deep CTR models. However, searching fine-grained regularization coefficients will involve additional burden, bringing higher requirements for efficient hyperparameter search.
- In comparison with grid search, HPO methods (e.g., PBT, AutoLRS, BO, BOHB, AutoL2, AutoOpt) perform better in most cases. This is because grid search restricts hyperparameters to discrete search space, while others explore in continuous space.
- Among the HPO methods, BO and BOHB fix the hyperparameters in each repeated training process, while AutoOpt schedules hyperparameters along with training adaptive, thus getting better performance. Compared with PBT, AutoLRS and AutoL2, our proposed AutoOpt employs the Bayesian optimization method to capture the relation between model states and performance. Besides, a multi-worker parallel training scheme is proposed to ensure a more robust training process. In all, AutoOpt achieves superior performance by adaptively scheduling fine-grained hyperparameters along with model training.

## 5.3 Effectiveness and Efficiency

In this section, we investigate the effectiveness and efficiency of AutoOpt compared with five baselines. Table 3 summarizes the computation cost of different methods measuring in GPU hours, where DeepFM is the used backbone model and the dataset is Criteo. From this, we have the following observations:

Firstly, Grid-Component costs mostly due to its large discrete search space, while Grid-Same reduces the cost with the same coefficient for all components (with worse prediction accuracy). Secondly, BO needs to repeat the procedure of training multiple times, which also costs a lot. BOHB introduces hyperband to reduce some hyperparameters that perform not well, thus requiring less search cost than BO. Finally, AutoOpt and AutoL2 are time-efficient because they schedule hyperparameters during training. Note that the reason why AutoL2 costs less is that it only introduces rule-based scheduling, while AutoOpt learns a model to predict optimal hyperparameters and employs a parallel training scheme to get better performance.

To further understand the trade-off between effectiveness and efficiency, we illustrate the prediction AUC and the training GPU hours of different methods with different backbone models in Figure 6. Points in the left upper area of the space are well performed in both effectiveness and efficiency.

From Figure 6 we can find that, although AutoL2 is the most efficient method, its predefined rule for scheduling prevents it from gaining better performance. Besides, BO has comparable performance on some models but costs too much on xDeepFM, which is the most complex model among backbone models. BOHB introduces the hyperband technique to reduce the computation cost but still costs more to gain performance compared with AutoOpt. Instead, AutoOpt achieves the best performance with relatively few GPU hours, showing superior effectiveness and efficiency. We attribute it to the superiority of the stage-wise scheduling algorithm and multi-worker parallel training scheme.

## 5.4 Ablation Study

Moreover, to present the effects of different optimization-related hyperparameters, we apply AutoOpt to the DCN backbone model. Specifically, we remove each scheduling hyperparameter from AutoOpt and fix it during the model training process. From Figure 7 we can find that searching all hyperparameters with AutoOpt can achieve the best performance. Besides, compared with other hyperparameters, scheduling the learning rate and regularization coefficient of deep components among the training procedure can obtain higher profits.

## 5.5 Hyper-Parameters Sensitivity

To study the effects of training epochs $K$ in each stage and the number of workers $N$ for AutoOpt, we conduct several hyperparameters studies on Avazu dataset with DeepFM as the backbone. In the left subfigure of Figure 8, we compare model performance with different training epochs $K$ in fixed training cost, i.e., $T * K = 50$

**Table 2: Overall performance comparison in test dataset of different hyperparameter search methods with several popular backbone models in three datasets. The best results in baseline methods are marked by underline, and the AutoOpt results are emphasized in bold. ⋆ represents significance level $p$-value$<0.05$ of comparing AutoOpt with the best baselines.**

| Dataset | Method | IPNN | | DeepFM | | DCN | | xDeepFM | |
|---|---|---|---|---|---|---|---|---|---|
| | | AUC | logloss | AUC | logloss | AUC | logloss | AUC | logloss |
| Criteo | LR-Decay | 0.79770 | 0.45298 | 0.79307 | 0.45711 | 0.78200 | 0.46662 | 0.79891 | 0.45184 |
| | Grid-Same | 0.80179 | 0.44994 | 0.80364 | 0.44833 | 0.80341 | 0.44896 | 0.80382 | 0.44889 |
| | Grid-Componet | 0.80285 | 0.44911 | 0.80479 | 0.44677 | 0.80471 | 0.44679 | 0.80506 | 0.44611 |
| | PBT | 0.80034 | 0.45546 | 0.80211 | 0.45301 | 0.80351 | 0.44840 | 0.80242 | 0.44872 |
| | AutoLRS | 0.79689 | 0.45367 | 0.79609 | 0.45437 | 0.79537 | 0.45503 | 0.79913 | 0.45169 |
| | BO | 0.80439 | 0.44678 | 0.80412 | 0.44693 | 0.80459 | 0.44680 | 0.80455 | 0.44678 |
| | BOHB | 0.80460 | 0.44659 | 0.80422 | 0.44687 | 0.80424 | 0.44687 | 0.80530 | 0.44598 |
| | AutoL2 | 0.80407 | 0.44752 | 0.80432 | 0.44684 | 0.80381 | 0.44741 | 0.80220 | 0.44980 |
| | AutoOpt | **0.80618**⋆ | **0.44520**⋆ | **0.80721**⋆ | **0.44432**⋆ | **0.80699**⋆ | **0.44450**⋆ | **0.80809**⋆ | **0.44347**⋆ |
| | % Improv. | 0.196% | 0.311% | 0.300% | 0.548% | 0.283% | 0.512% | 0.346% | 0.562% |
| Avazu | LR-Decay | 0.76836 | 0.38634 | 0.76401 | 0.38877 | 0.75544 | 0.39328 | 0.76588 | 0.38772 |
| | Grid-Same | 0.76525 | 0.38816 | 0.76646 | 0.38773 | 0.76465 | 0.38857 | 0.77016 | 0.38581 |
| | Grid-Component | 0.76649 | 0.38787 | 0.76946 | 0.38732 | 0.76662 | 0.38764 | 0.77382 | 0.38498 |
| | PBT | 0.77380 | 0.38330 | 0.77107 | 0.38430 | 0.76855 | 0.38608 | 0.77155 | 0.38392 |
| | AutoLRS | 0.76565 | 0.38802 | 0.76350 | 0.38921 | 0.76322 | 0.38921 | 0.76753 | 0.38685 |
| | BO | 0.77449 | 0.38346 | 0.77321 | 0.38417 | 0.77324 | 0.38388 | 0.77410 | 0.38312 |
| | BOHB | 0.77478 | 0.38333 | 0.77332 | 0.38364 | 0.77265 | 0.38395 | 0.77390 | 0.38351 |
| | AutoL2 | 0.77236 | 0.38427 | 0.77153 | 0.38449 | 0.77084 | 0.38483 | 0.76940 | 0.38737 |
| | AutoOpt | **0.77614**⋆ | **0.38242**⋆ | **0.77515**⋆ | **0.38310**⋆ | **0.77426**⋆ | **0.38316**⋆ | **0.77489**⋆ | **0.38260**⋆ |
| | % Improv. | 0.175% | 0.237% | 0.236% | 0.140% | 0.131% | 0.187% | 0.102% | 0.135% |
| KDD12 | LR-Decay | 0.79154 | 0.16011 | 0.78535 | 0.16316 | 0.78215 | 0.16529 | 0.79759 | 0.15947 |
| | Grid-Same | 0.78241 | 0.16291 | 0.78254 | 0.16284 | 0.77527 | 0.16491 | 0.78395 | 0.16288 |
| | Grid-Component | 0.78385 | 0.16265 | 0.78324 | 0.16234 | 0.78428 | 0.16256 | 0.79290 | 0.16037 |
| | PBT | 0.77014 | 0.16619 | 0.78130 | 0.16565 | 0.77327 | 0.16234 | 0.80042 | 0.15823 |
| | AutoLRS | 0.78778 | 0.16136 | 0.78572 | 0.16353 | 0.78266 | 0.16224 | 0.79762 | 0.15906 |
| | BO | 0.79127 | 0.16057 | 0.79228 | 0.15987 | 0.79075 | 0.16040 | 0.80069 | 0.15812 |
| | BOHB | 0.79162 | 0.16038 | 0.79149 | 0.15992 | 0.79826 | 0.15892 | 0.80093 | 0.15811 |
| | AutoL2 | 0.78584 | 0.16186 | 0.78497 | 0.16202 | 0.78523 | 0.16196 | 0.77926 | 0.16588 |
| | AutoOpt | **0.79552**⋆ | **0.15931**⋆ | **0.79486**⋆ | **0.15943**⋆ | **0.80261**⋆ | **0.15803**⋆ | **0.80380**⋆ | **0.15720**⋆ |
| | % Improv. | 0.492% | 0.667% | 0.325% | 0.275% | 0.544% | 0.560% | 0.358% | 0.575% |

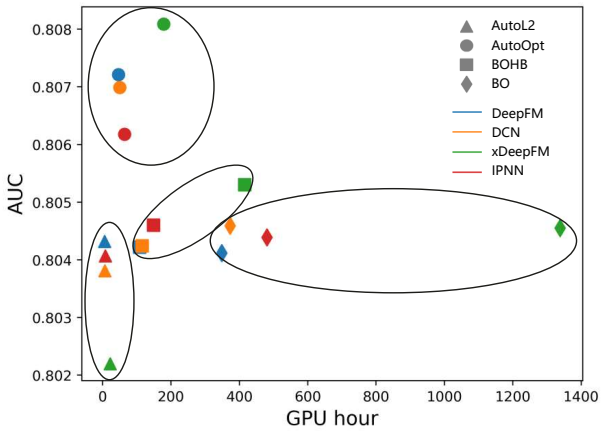


**Figure 6: Effectiveness and efficiency comparison of various methods with mainstream deep models in Criteo dataset.**

and the number of workers is $N = 8$. The results show that the model with $K = 2$ increases faster in the early training process since it adjusts hyperparameters more frequently. However, it does not converge to the best performance, because of insufficient data for

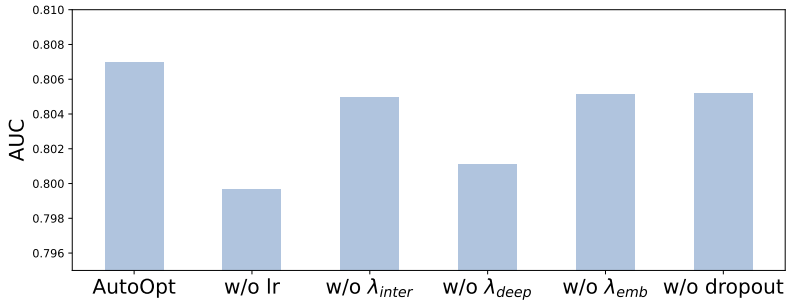


**Figure 7: Ablation study experiments on Criteo dataset by removing different scheduling hyperparameters.**

**Table 3: Search cost on Criteo dataset (measured in GPU hour)**

| Grid-Same | Grid-Componet | BO | BOHB | AutoL2 | AutoOpt |
|---|---|---|---|---|---|
| 402.6 | 3650.7 | 341.6 | 118.3 | 5.2 | 43.1 |

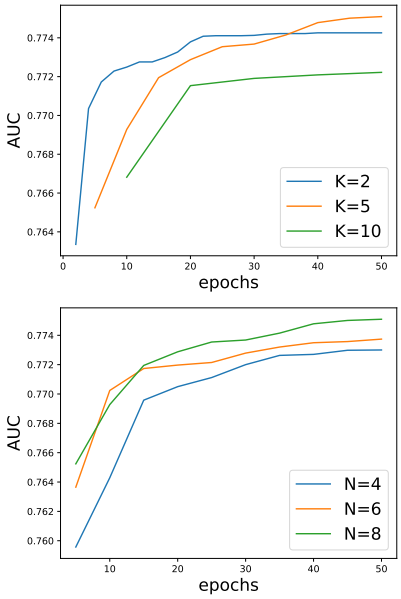


**Figure 8: The effects of training epochs $K$ in each stage and the number of workers $N$. The X-axis indicates the accumulated epochs.**

modeling performance in each stage. Conversely, the model with $K = 10$ rarely adjusts the coefficients in the training but converges quickly to low accuracy. We find that setting $K = 5$ epochs enables the model to get the best performance.

In the right subfigure of Figure 8, we compare model performance with the different numbers of workers $N$. We can get that the model with more workers has better performance, but it takes nearly linear growth of computation resources. Hence, we set the worker number as $N = 8$ due to its good trade-off between efficiency and model performance.

## 5.6 Case Study

To illustrate the scheduling process of hyperparameters with training, we perform a case study to investigate the scheduling of AutoOpt. Figure 9 presents the learning rate and coefficient of deep component for various backbone models. As can be observed, although the learning rates in backbones are different, they all decay to a relatively lower value compared to the initial period, which is consistent with the statement in previous work [27]. The rationale behind this is that when the model tends to converge, a small learning rate is more helpful to get further improvement. AutoOpt schedules the $L_2$ coefficient without a notable pattern, which further demonstrates that it is difficult to set the scheduling of the regularization coefficient in advance either by a predefined rule or by grid search. This observation further verifies the necessity of our method.

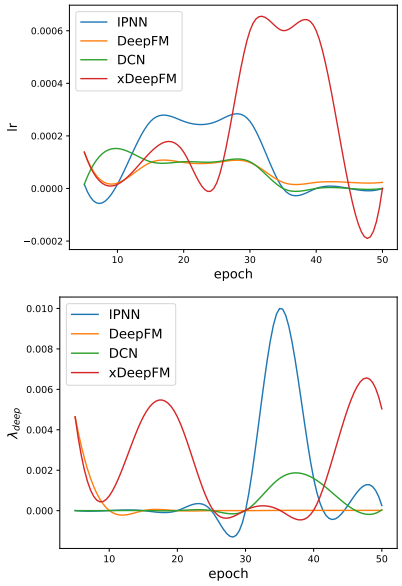


**Figure 9: Learning rate and $L_2$ coefficient of deep component scheduled by AutoOpt on Avazu dataset.**

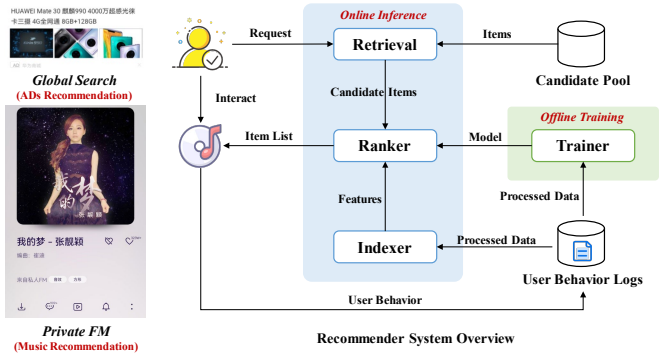


**Figure 10: The scenarios of *Global Search* and *Private FM*, and overview of a mainstream Recommender System.**

# 6 ONLINE EXPERIMENT

## 6.1 Scenario Description & System Overview

We deploy the AutoOpt in a mainstream network information service platform to enable adaptive hyperparameters adjustment for deep CTR models, serving tens of millions of daily active users. The A/B testing is conducted in two scenarios: 1) An advertising display scenario named "*Global Search*"; 2) A personalized song recommendation scenario named "*Private FM*". The display scenarios are shown in the left subgraph of Figure 10.

We briefly describe the industrial recommender system, which is depicted in the right subgraph of Figure 10. The system overview consists of two core modules: Online Inference stage and Offline Training stage. When a user arrives, a request with the user's attributes and contextual features is sent to the online service. Then the online *Retrieval* module is triggered and retrieves candidate items (ads or music) from the candidate pool. An *Indexer* extracts features of the user, candidate items, and the context, and then constructs instances for online serving. Afterward, a *Ranker* leverages these instances and the model that is trained periodically by an offline *Trainer* module to compute pCTR scores. Finally, an item list is presented to the user, which is sorted by a pre-defined ranking function. As for the offline training stage, the *Trainer* module utilizes the newly generated user behavior logs to train deep CTR models, which will be pushed for online serving periodically. Our proposed AutoOpt is deployed in the offline *Trainer* module and provides automatic hyperparameter adjustment capability for various deep CTR models. It is obvious that AutoOpt is easy to deploy and has a wide range of compatibility. More importantly, the online training procedure is imperceptible, avoiding online service modification.

## 6.2 Experimental Setting

The online A/B test is conducted for three weeks on both the ads and music recommendation scenarios, where tens of millions of daily active users are served and generate plenty of user log events. The compared baseline model online is denoted as $\mathcal{M}_{Base}$, which is a carefully-tuned deep CTR model. Based on the same model architecture as $\mathcal{M}_{Base}$, we deploy the AutoOpt to automatically search optimal learning rate, component-level $L_1$ and $L_2$ coefficients, dropout ratio, named as $\mathcal{M}_{AutoOpt}$. Both $\mathcal{M}_{Base}$ and $\mathcal{M}_{AutoOpt}$ are trained over the same latest user behavior logs. For online ads (or music) serving, 10% (or 25%) of the users are randomly selected as the experimental group and served by the AutoOpt model $\mathcal{M}_{AutoOpt}$, while another 10% (or 25%) of the users are in the control group and served by the baseline model $\mathcal{M}_{Base}$.

## 6.3 Online Results

For online advertising recommendation, two commonly-used advertising evaluation metrics, *i.e.*, **click-through rate (CTR)** (total number of clicks divided by the total number of impressions) and **effective cost per mile (eCPM)** (total revenue divided by the total number of thousand ad impressions), are used to evaluate the performance of different deployed models. Besides, for music recommendation, we introduce two widely-used music evaluation metrics, *i.e.*, **per capita playback times (PCPT)** (total playback times divided by the number of users) and **per capita playback duration (PCPD)** (total playback duration divided by the number of users). The online A/B testing results are shown in Table 4. Compared with the control group, $\mathcal{M}_{AutoOpt}$ upgrades the average CTR and eCPM by **3.6%** and **1.4%** for ads recommendation; upgrades PCPT and PCPD by **5.3%** and **4.9%** for music recommendation, which demonstrates the effectiveness of the AutoOpt method. Besides, the training time of $\mathcal{M}_{AutoOpt}$ is increased by only **18%**, which is acceptable in the industry. Additionally, the inference time that is more concerned with the CTR prediction does not increase. Moreover, with AutoOpt integrated, the tedious hyperparameter adjustment process can be omitted, thereby saving significant labor costs and bringing significant business improvement.

**Table 4: Online improvement of $\mathcal{M}_{AutoOpt}$ compared with baseline $\mathcal{M}_{Base}$. The improvement is calculated by A/B - A/A.**

| Global Search (ads) | CTR | eCPM |
|---|---|---|
| | +3.6% | +1.4% |
| Private FM (music) | PCPT | PCPD |
| | +5.3% | +4.9% |

# 7 CONCLUSION

We propose AutoOpt to automatically schedule and optimize hyperparameters for deep CTR models, making the optimization process

more stable and efficient. We formulate the training process as several consecutive stages where each stage consists of three main phases, where stage-wise hyperparameters are proposed. Moreover, to improve the stability and efficiency of hyperparameter exploration, a multi-worker parallel training system is also proposed. Extensive experiments on both offline benchmark CTR datasets and online tests demonstrate that AutoOpt can improve the performance and reduce the search cost of several mainstream deep CTR models.

## ACKNOWLEDGMENTS

We thank MindSpore [1] for the partial support of this work, which is a new deep learning computing framework.

## REFERENCES

[1] 2020. MindSpore. https://www.mindspore.cn/
[2] Weijie Bian, Kailun Wu, Lejian Ren, Qi Pi, Yujing Zhang, Can Xiao, Xiang-Rong Sheng, Yong-Nan Zhu, Zhangming Chan, Na Mou, Xinchen Luo, Shiming Xiang, Guorui Zhou, Xiaoqiang Zhu, and Hongbo Deng. 2022. CAN: Feature Co-Action Network for Click-Through Rate Prediction. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining* (Virtual Event, AZ, USA) *(WSDM '22)*. Association for Computing Machinery, New York, NY, USA, 57–65. https://doi.org/10.1145/3488560.3498435
[3] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2015. Simple and Scalable Response Prediction for Display Advertising. *ACM Trans. Intell. Syst. Technol.* 5, 4 (dec 2015), 61.
[4] Karansingh Chauhan, Shreena Jani, Dhrumin Thakkar, Riddham Dave, Jitendra Bhatia, Sudeep Tanwar, and M. Obaidat. 2020. Automated Machine Learning: The New Wave of Machine Learning. In *International Conference on Innovative Mechanisms for Industry Applications*. 205–212.
[5] Hung-Hsuan Chen and Pu Chen. 2019. Differentiating Regularization Weights–A Simple Mechanism to Alleviate Cold Start in Recommender Systems. *ACM Trans. Knowl. Discov. Data* 13, 1, Article 8 (Jan. 2019).
[6] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.
[7] Yihong Chen, B. Chen, Xiangnan He, Chen Gao, Y. Li, Jian-Guang Lou, and Yue Wang. 2019. $\lambda$Opt: Learn to Regularize Recommender Models in Finer Levels. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 978–986.
[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 7–10.
[9] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 1437–1446.
[10] Bruno G. Galuzzi, Ilaria Giordani, Antonio Candelieri, Riccardo Perego, and Francesco Archetti. 2020. Hyperparameter optimization for recommender systems through Bayesian optimization. *Comput. Manag. Sci.* 17, 4 (2020), 495–515. https://doi.org/10.1007/s10287-020-00376-3
[11] Agathe Girard, Carl Edward Rasmussen, Joaquin Quiñonero Candela, and Roderick Murray-Smith. 2002. Gaussian Process Priors with Uncertain Inputs-Application to Multiple-Step Ahead Time Series Forecasting. In *NeurIPS*.
[12] Huifeng Guo, Bo Chen, Ruiming Tang, Weinan Zhang, Zhenguo Li, and Xiuqiang He. 2021. An Embedding Learning Framework for Numerical Features in CTR Prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2910–2918. https://doi.org/10.1145/3447548.3467077
[13] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*.
[14] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. 1–9.
[15] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).
[16] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2008. A Practical Guide to Support Vector Classication.
[17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).
[18] Yuchen Jin, Tianyi Zhou, Liangyu Zhao, Yibo Zhu, Chuanxiong Guo, Marco Canini, and Arvind Krishnamurthy. 2021. Autolrs: Automatic learning-rate schedule by bayesian optimization on the fly. *arXiv preprint arXiv:2105.10762* (2021).
[19] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2387–2397.
[20] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-Aware Factorization Machines for CTR Prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 43–50.
[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[22] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[23] Lang Lang, Zhenlong Zhu, Xuanye Liu, Jianxin Zhao, Jixing Xu, and Minghui Shan. 2021. Architecture and Operation Adaptive Network for Online Recommendations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3139–3149.
[24] Aitor Lewkowycz and Guy Gur-Ari. 2020. On the training dynamics of deep networks with $L_2$ regularization. In *Advances in Neural Information Processing Systems*, Vol. 33. 4790–4799.
[25] Jianxun Lian, Xiaohua Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guang zhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1754–1763.
[26] Bin Liu, Niannan Xue, Huifeng Guo, Ruiming Tang, Stefanos Zafeiriou, Xiuqiang He, and 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* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 199–208. https://doi.org/10.1145/3397271.3401082
[27] Ilya Loshchilov and Frank Hutter. 2016. SGDR:Stochastic Gradient Descent with Restarts. In *ICLR*.
[28] Fuyuan Lyu, Xing Tang, Huifeng Guo, Ruiming Tang, Xiuqiang He, Rui Zhang, and Xue Liu. 2022. Memorize, Factorize, or be Naive: Learning Optimal Feature Interaction Methods for CTR Prediction. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1450–1462. https://doi.org/10.1109/ICDE53745.2022.00113
[29] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. https://doi.org/10.48550/ARXIV.2208.04482
[30] Hao Peng, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. 2015. A Comparative Study on Regularization Strategies for Embedding-based Neural Networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2106–2111.
[31] 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. *ACM Trans. Inf. Syst.* 37, 1, Article 5 (2018).
[32] C. Rasmussen and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. The MIT Press.
[33] Steffen Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000.
[34] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting Clicks: Estimating the Click-through Rate for New Ads. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Alberta, Canada) *(WWW '07)*. Association for Computing Machinery, New York, NY, USA, 521–530.
[35] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. 2002. Methods and Metrics for Cold-Start Recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 253–260.
[36] Jasper Snoek, H. Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NeurIPS*.
[37] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. 2020. Towards automated neural interaction discovery for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 945–955.
[38] Jianhui Sun, Ying Yang, Guangxu Xun, and Aidong Zhang. 2021. A Stagewise Hyperparameter Scheduler to Improve Generalization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY,

USA, 1530–1540. https://doi.org/10.1145/3447548.3467287
[39] Yang Sun, Junwei Pan, Alex Zhang, and Aaron Flores. 2021. FM2: Field-Matrixed Factorization Machines for Recommender Systems. In *Proceedings of the Web Conference 2021*. 2828–2837.
[40] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*. 12:1–12:7.
[41] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. 2019. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878* (2019).
[42] Yun Yue, Yongchao Liu, Suo Tong, Minghao Li, Zhen Zhang, Chunyang Wen, Huanjun Bao, Lihong Gu, Jinjie Gu, and Yixiang Mu. 2021. Adaptive Optimizers with Sparse Group Lasso for Neural Networks in CTR Prediction. In *Machine Learning and Knowledge Discovery in Databases. Research Track (ECML-PKDD 2021)*. Springer International Publishing, Cham, 314–329.
[43] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data. In *European Conference on Information Retrieval*. 45–57.
[44] Weinan Zhang, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He. 2021. Deep Learning for Click-Through Rate Estimation. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. 4695–4703.
[45] Pengyu Zhao, Kecheng Xiao, Yuanxing Zhang, Kaigui Bian, and Wei Yan. 2021. AMEIR: Automatic Behavior Modeling, Interaction Exploration and MLP Investigation in the Recommender System.. In *IJCAI*. 2104–2110.
[46] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2021. AutoDim: Field-aware Embedding Dimension Searchin Recommender Systems. In *WWW '21: The Web Conference 2021*. ACM / IW3C2, Slovenia, 3015–3022.
[47] Chenxu Zhu, Bo Chen, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2021. AIM: Automatic Interaction Machine for Click-Through Rate Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2021).