Approximate Nearest Neighbor Search under Neural Similarity Metric for Large-Scale Recommendation

Rihan Chen
Bin Liu
Han Zhu
rihan.crh@alibaba-inc.com
Alibaba Group
Beijing, China

Yaoxuan Wang
Qi Li
Buting Ma
Qingbo Hua
Alibaba Group
Beijing, China

Jun Jiang
Yunlong Xu
Hongbo Deng
Bo Zheng
Alibaba Group
Beijing, China

ABSTRACT

Model-based methods for recommender systems have been studied extensively for years. Modern recommender systems usually resort to 1) representation learning models which define user-item preference as the distance between their embedding representations, and 2) embedding-based Approximate Nearest Neighbor (ANN) search to tackle the efficiency problem introduced by large-scale corpus. While providing efficient retrieval, the embedding-based retrieval pattern also limits the model capacity since the form of user-item preference measure is restricted to the distance between their embedding representations. However, for other more precise user-item preference measures, e.g., preference scores directly derived from a deep neural network, they are computationally intractable because of the lack of an efficient retrieval method, and an exhaustive search for all user-item pairs is impractical.

In this paper, we propose a novel method to extend ANN search to arbitrary matching functions, e.g., a deep neural network. Our main idea is to perform a greedy walk with a matching function in a similarity graph constructed from all items. To solve the problem that the similarity measures of graph construction and user-item matching function are heterogeneous, we propose a pluggable adversarial training task to ensure the graph search with arbitrary matching function can achieve fairly high precision. Experimental results in both open source and industry datasets demonstrate the effectiveness of our method. The proposed method has been fully deployed in the Taobao display advertising platform and brings a considerable advertising revenue increase. We also summarize our detailed experiences in deployment in this paper.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Personalization**.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557098

KEYWORDS

Approximate Nearest Neighbor Search, Model-based Retrieval, Recommender Systems

ACM Reference Format:
Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, and Bo Zheng. 2022. Approximate Nearest Neighbor Search under Neural Similarity Metric for Large-Scale Recommendation. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557098

1 INTRODUCTION

Constantly growing amount of available information has posed great challenges to modern recommenders. To deal with the information explosion, modern recommender system is usually designed with a multi-stage cascade architecture that mainly consists of candidate generation and ranking. In the candidate generation stage, also known as matching, thousands of targets are retrieved from a very large corpus, and then, in the ranking stage, these retrieved targets are ranked according to the user's preference. Notably, given the constraints of computational resources and latency in real-world systems, candidate generation cannot be solved by sequentially scanning the entire corpus while facing a large-scale corpus.

To bypass the prohibitive computational cost of scanning the entire corpus, embedding-based retrieval (EBR) has prevailed in recommender systems for years due to its simplicity and efficiency [15, 18]. However, EBR is insufficient to model the complex structure of user-item preferences. Many works have already shown that more complex models usually generalize better [13, 24, 36]. And researchers have striven to develop techniques to tackle the large-scale retrieval problem with more complex models as well. To overcome computation barriers and benefit from arbitrarily advanced models, the idea of regularizing the total computational cost through an index has recently been presented. These methods [10, 37–39] typically have a learnable index and follow the Expectation Maximization (EM) type optimization paradigm, updating between deep model and index alternatively. As a consequence, the deep model, together with beam search, can be leveraged to retrieve relevant items from a large corpus in a sub-linear complexity w.r.t. corpus size. Even though these end-to-end methods can introduce a deep model to large-scale retrieval, there are two aspects that should not be ignored: 1) the joint training of index and model for large-scale data

necessitates a costly training budget in terms of both training time and computational resources; 2) the existence of index structure's internal nodes, such as non-leaf nodes in TDMs [37–39] and path nodes in DR [10], makes it difficult to utilize side-information from items.

This work tackles the aforementioned problems by solving large-scale retrieval with an arbitrarily advanced model in a lightweight manner, called Neural Approximate Nearest Neighbour Search (NANN). More specifically, we leverage the deep model as a greedy walker to explore a similarity graph constructed after model training. The joint training budget of the end-to-end methods can be greatly released by following the decoupled paradigm. Besides, the similarity graph that the deep model traverses contains no internal nodes, which facilitates the usage of side information from candidate items. To improve the efficiency and effectiveness of graph search, we creatively come up with both a heuristic retrieval method and a auxiliary training task in our NANN framework. The main contributions of our paper are summarized as follows:

- We present a unified and lightweight framework that can introduce arbitrarily advanced models as the matching function to large-scale ANN retrieval. The basic idea is to leverage similarity graph search with the matching function.
- To make the computational cost and latency controllable in graph search, we propose a heuristic retrieval method called Beam-retrieval, which can reach better results with fewer computations. And we also propose an auxiliary adversarial task in model training, which can greatly mitigate the effect of heterogeneity between similarity measures and improve the retrieval quality.
- We conduct extensive experiments on both a publicly accessible benchmark dataset and a real industry dataset, which demonstrate the proposed NANN is an excellent empirical solution to ANN search under neural similarity metric. Besides, NANN has been fully deployed in the Taobao display advertising platform and contributes 3.1% advertising revenue improvements.
- We describe in detail the hands-on deployment experiences of NANN in Taobao display advertising platform. The deployment and its corresponding optimizations are based on the Tensorflow framework [1]. We hope that our experiences in developing such a lightweight yet effective large-scale retrieval framework will be helpful to outstretch NANN to other scenarios with ease.

2 RELATED WORK

Hereafter, let $\mathcal{V}$ and $\mathcal{U}$ denote the item set and the user set. In recommendation, we strive to retrieve a set of relevant items $\mathcal{B}_u$ from a large-scale corpus $\mathcal{V}$ for each user $u \in \mathcal{U}$. Mathematically,

$$\mathcal{B}_u = \underset{v \in \mathcal{V}}{\text{argTopk}}\, s_u(\mathbf{e}_v) \tag{1}$$

where $s_u(\mathbf{e}_v)$ is the similarity function between user u and item v. $\mathbf{e}_v$ is the mathematical representation for item v, which is the only variable w.r.t $s_u(.)$ when search on graph-based index for user u.

Search on graph. Search on graph popularized by its exceptional efficiency, performance, and flexibility (in terms of similarity function) is a fundamental and powerful approach for NNS. The theoretical foundation to search on graph is Voronoi diagram and its straight-line dual, the s-Delaunay graph (also called s-Delaunay triangulation) defined by similarity function $s_u(\mathbf{e}_v)$ [2, 9]. Previous work [23] has shown that $s_u(\mathbf{e}_v) = -||v - u||_2$ can find the exact solution to Equation (1), when $k = 1$ and $u, v \in \mathbb{R}^d$, by certain greedy walk on the s-Delaunay graph constructed from $\mathcal{V}$. More generally, many existing works attempt to extend the conclusion to non-metric cases, such as inner product [3, 25, 27, 28], Mercer kernel [6, 7] and Bregman divergence [4]. In addition, researchers also set foot in approximating the s-Delaunay graph as the construction of a perfect s-Delaunay graph with a large corpus is infeasible. For example, the k-Nearest Neighbor (kNN) graph turns out to actually be a subgraph of the Delaunay triangulation. Moreover, Navigable Small World (NSW) [20] is proposed to greatly optimize both graph construction and search process. On top of that, Hierarchical NSW (HNSW) [21], the variant of NSW, provides state-of-the-art for NNS, which incrementally builds a multi-layer structure from proximity graphs and retrieve a cascade of graphs in top-down approach. NANN will resort to HNSW, although other graph-based NNS methods can also work.

Deep model-based retrieval. Model-based, especially deep model-based methods have been an active topic in large-scale retrieval recently. In recommendation, many works focus on an end-to-end fashion to simultaneously train index and deep model. Tree-based methods, including TDM [38], JTM [37] and BSAT [39], build its index as a tree structure and model user interests from coarse to fine. Deep retrieval (DR) [10] encodes all candidate items with learnable paths and train the item paths along with the deep model to maximize the same objective. These approaches traverse their index to predict user interests and achieve sub-linear computational complexity w.r.t corpus size by beam search. However, these methods usually require additional internal nodes to parametrize the learnable index, which imposes difficulties in using side information of items. Moreover, additional model parameters and training time have to be paid for these end-to-end manners due to the existence of a learnable index and EM-type training paradigm.

Search on the graph with deep model A few works have already tried to extend the similarity function to deep neural networks. The closest work to ours is SL2G [31] which constructs the index graph by $l2$ distance and traverses the post-training graph with deep neural networks. However, their approach can be only generalized to the $s_u(\mathbf{e}_v)$ with convexity or quasi-convexity. For the non-convex similarity function (most common case for deep neural network), they apply SL2G directly without adaption. Another work [22] defines the index graph without similarity for item pairs. They exploit the idea that relevant items should have close $s(v, u)$ for the same user and represent a candidate item by a subsample of $\{s_u(\mathbf{e}_v)|j = 1, \ldots, m\}$. However, it is difficult to sample a representative set in practice, especially for large-scale corpus $\mathcal{V}$.

3 METHODOLOGY

In this section, we firstly give a general framework about EBR and model-based retrieval in Section 3.1, including model architecture and training paradigm. Then, we introduce the similarity graph construction and graph-based retrieval method respectively for the

proposed NANN in Section 3.2 and Section 3.3. Given these preliminary concepts, we accordingly introduce the pluggable adversarial training task and demonstrate how it can eliminate the gap of similarity measures between graph construction and model-based matching function in Section 3.4.

3.1 General Framework

3.1.1 Review Embedding Based Retrieval. Our proposed method can be generally deemed as an extension of the EBR framework where we generalize the simple similarity metrics to arbitrary neural ones. Therefore, we briefly review the EBR framework for clarity.

EBR is designed with a two-sided model architecture where one side is to encode the user profile and behaviour sequence, and the other side is to encode the item. Mathematically,

$$\mathbf{e}_u = \mathrm{NN}_u(\mathbf{f}_u),\ \mathbf{e}_v = \mathrm{NN}_v(\mathbf{f}_v). \tag{2}$$

where two deep neural networks NN_u and NN_v (i.e., the user and the item network) encode the inputs of $\mathbf{f}_u$ and $\mathbf{f}_v$ to the dense vectors $\mathbf{e}_u \in \mathbb{R}^d$ and $\mathbf{e}_v \in \mathbb{R}^d$ separately. And the user-item preference forms as the inner product of the semantic embedding, i.e. $\mathbf{e}_u^T\mathbf{e}_v$. The candidate sampling based criterion such as Noise Contrastive Estimation (NCE) [12] and Sampled-softmax [16] are usually used to train the EBR models due to the computational difficulty to evaluate partition functions by summing over the entire vocabulary of large corpus. At inference stage, EBR model is usually decomposed into a user embedding model and an item embedding model. And we obtain user embedding $\mathbf{e}_u$ with real-time prediction (because $\mathbf{f}_u$ usually contains real-time features) and item embedding by offline precomputation.

3.1.2 Model Architecture. Compared to the traditional EBR method in large-scale retrieval, NANN greatly outstretches the model capacity by more complex architecture with user network, target attention network, and item network akin to a standard CTR prediction model, as shown in Figure 1. In other words, we substitute the inner product $\mathbf{e}_u^T\mathbf{e}_v$ with a more general and expressive $s_u(\mathbf{e}_v)$. The generalized form $s_u(\mathbf{e}_v)$ with deep neural network, in turn, poses both theoretical and practical challenges to us: 1) how to generalize the search on the graph-based index to any non-linear and non-convex $s_u(\mathbf{e}_v)$ reasonably; 2) how to integrate the graph-based index with complex deep model and deploy the computation-intensive retrieval framework in a lightweight and efficient way.

3.1.3 Training. Same with EBR, we reduce the computationally intractable problem to the problem of estimating the parameters of a binary classifier by NCE. The positive samples come from the true distribution that user u engages with item v, while the negative samples are drawn from a "noise" distribution $q(v)$, e.g., the unigram distribution over $v \in \mathcal{V}$. We denote the corresponding loss function as $\mathcal{L}_{NCE}$. Moreover, we extend the search on the graph-based index to any metric $s_u(\mathbf{e}_v)$ by using an auxiliary task with the loss denoted by $\mathcal{L}_{AUX}$ (details are in Section 3.4). Hence, the overall objective is

$$\mathcal{L}_{all} = \mathcal{L}_{NCE} + \mathcal{L}_{AUX}. \tag{3}$$

3.1.4 Search on post-training similarity graph. The graph-based index is built from the precomputed item embedding $\mathbf{e}_v$ extracted from item network NN_v. In the prediction stage, we traverse the similarity graph in a way that is tailored to both real-world systems and arbitrary $s_u(\mathbf{e}_v)$.

3.2 Graph Construction

Search on similarity graphs was originally proposed for metric spaces and extended to the symmetric non-metric scenarios, e.g, Mercer kernel and Maximum Inner Product Search (MIPS). The $s_u(\mathbf{e}_v)$ can be also generalized to the certain asymmetric case, i.e. Bregman divergence, by exploiting convexity in place of triangle inequality [4]. However, s-Delaunay graph with arbitrary $s_u(\mathbf{e}_v)$ is not guaranteed to exist or be unique. Furthermore, to construct such s-Delaunay graphs from the large-scale corpus are even computationally prohibitive for both exact and approximate ones. Hence, we follow the way of SL2G [31] to simplify this problem by building the graph index with the item embedding $\mathbf{e}_v$. The graph is defined with $l2$ distance among $\mathbf{e}_v$ and agnostic to $s_u(\mathbf{e}_v)$. In practice, we build the HNSW graph directly, which is claimed a proper way to approximate the Delaunay graph defined on $l2$ distance.

3.3 Online Retrieval

We equip the original HNSW with beam search and propose a Beam-retrieval to handle the online retrieval in production. The search process of HNSW traverses a hierarchy of proximity graphs in a layer-wise and top-down way, as shown in Algorithm 1. The original HNSW retrieval algorithm referred to as HNSW-retrieval for convenience, employs simple greedy searches where $ef_l (l > 0)$ in Algorithm 1 is set to 1 at the top layers and assigns a larger value to ef_0 to ensure retrieval performance at the ground layer. However, the HNSW-retrieval is practically insufficient to tackle large-scale retrieval in real-world recommender systems since it suffers from the following deficiencies: 1) the subroutine SEARCH − LAYER in HNSW-retrieval explores the graph in a while-loop, which makes the online inference's computation and latency uncontrollable; 2) the traversal with simple greedy search is more prone to stuck into local optimum, especially for our case where $s_u(\mathbf{e}_v)$ is usually non-convex. Hence, we reform the SEARCH − LAYER subroutine in HNSW-retrieval according to Algorithm 2. We firstly replace the while-loop with a for-loop to control the prediction latency and the amount of candidates v to evaluate. Despite having an early-stopping strategy, the for-loop can still guarantee the retrieval performance, shown in Figure 6. We secondly break the limits on $ef_l (l > 0)$ and enlarge it at top layers to utilize batch computing. Traversal with multiple paths is equivalent to beam search on the similarity graph, which is proved more efficient than the original version demonstrated in Figure 4.

3.4 Search with Arbitrary Neural Metric

3.4.1 Motivation. When facing arbitrarily models, triangle inequality, symmetry and convexity can no longer be exploited to validate the rationality of similarity graph search with $s_u(\mathbf{e}_v)$. In practice, the reaction of $s_u(\mathbf{e}_v)$ to small perturbation of $\mathbf{e}_v$ is highly uncertain, e.g., $s_u(e_v)$ may fluctuate drastically when $\mathbf{e}_v$ is slightly perturbed. Intuitively, this uncertainty plagues the retrieval perforamnce especially when the similarity metrics used in graph construction stage ($l2$ distance among $\mathbf{e}_v$) and retrieval stage ($s_u(\mathbf{e}_v)$) are highly heterogeneous, shown in Table 3. And in this work, we show that retrieval

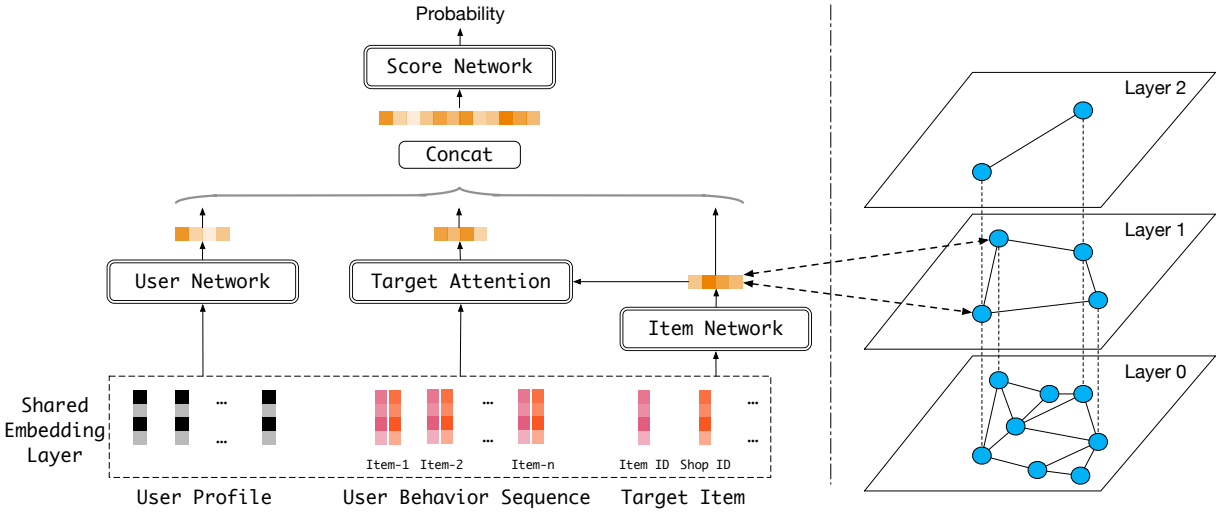


Figure 1: General Framework. In the left part, the deep model contains three basic branches, user network, target attention network, and item network. The user network is responsible for modeling the user's profile. And we adopt the attention mechanism to flexibly learn the interaction between user behavior sequence and target item. The information of items is learned through the item network. In the right part, we approximate the Delaunay graph defined on $l2$ distance among $\mathbf{e}_v$ (the output of item network) following HNSW. Note that the online inference starts from $\mathbf{e}_v$ for item network branch.

Algorithm 1: K-NN-SEARCH(s, G, u, K, ef)

Input: model s, multi-layer HNSW graph G, query user u, number of nearest neighbors to return K, size of the dynamic candidate list and number of steps to search for each layer $\{ef_l\}, \{T_l\}$
Output: K results with largest score $s_u(\mathbf{e}_v)$ to user u
$W \leftarrow \emptyset$ // the set of the current nearest elements
$ep \leftarrow$ get enter points from HNSW graph G
$L \leftarrow$ level of ep // the top layer of HNSW graph
for $l \leftarrow L \cdots 1$ **do**
  $W \leftarrow$ SEARCH-LAYER($s, u, ep, ef_l, l = l, T_l$)
  $ep \leftarrow W$
$W \leftarrow$ SEARCH-LAYER($s, u, ep, ef_0, l = 0, T_0$)
return $argtopk_{v \in W} f(u, v)$

Algorithm 2: SEARCH-LAYER(s, u, ep, ef_c, l_c, T_c)

Input: enter points ep, current layer number l_c, number of steps to search in this layer T_c
Output: ef_c results with largest score $s_u(\mathbf{e}_v)$ to user u
$S \leftarrow ep$ // set of visited elements
$C \leftarrow ep$ // set for candidates
$W \leftarrow ep$ // dynamic list of results
for $t \leftarrow 1 \ldots T_c$ **do**
  $N \leftarrow$ union of neighbors at layer l_c of all items in C
  $N \leftarrow N - S$ // pruning visited nodes
  $S \leftarrow S \cup N$ // mark as visited
  $W \leftarrow argTopK_{v \in W \cup N} s_u(\mathbf{e}_v)$
  $C \leftarrow W \cap N$ // new candidates
return W

performance can be empirically augmented if we intentionally bias $s_u(\mathbf{e}_v)$ to avoid uncertainty w.r.t $\mathbf{e}_v$.

Our philosophy is based upon an analogy to identifying the local optimum of a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ along with a certain direction. Suppose that the solution to $\min -s_u(e_v)$ is an arbitrary vector defined in R^d, gradient descent and coordinate descent are commonly used to find the local optimum. And we claim that the graph search is analogous to block coordinate descent, of which the update direction is governed by graph structure and top-k procedure instead of gradients. Hence, given the above, we can interpret the uncertainty of $s_u(\mathbf{e}_v)$ w.r.t $\mathbf{e}_v$ as analogous to the flat/sharpness of loss landscape in gradient-based optimization. Although disputable, it is widely thought that "flat minimal" usually generalize better compared to "sharp minimal" [19, 33] because of their robustness to small perturbation of inputs. Earlier works have attempted to change the optimization algorithm to favor flat minimal and find "better" regions [5, 8, 14]. Inspired by these works, we leverage the adversarial training [11, 29, 30, 33, 34] to both mitigate the uncertainty and improve the robustness of arbitrary $s_u(\mathbf{e}_v)$ w.r.t $\mathbf{e}_v$.

3.4.2 Adversarial Gradient Method. Generally speaking, we resort to the adversarial gradient method and introduce flatness into $s_u(\mathbf{e}_v)$ in an end-to-end learning-based method [33].

To achieve the robustness of deep neural networks by the defense against adversarial examples has been widely applied to various computer vision tasks in recent years [11, 29, 30, 34]. Adversarial examples refer to normal inputs with crafted perturbations which are usually human-imperceptible but can fool deep neural networks maliciously. The adversarial training utilized in our work is one of the most effective approaches [26, 32] defending against adversarial examples for deep learning. More specifically, we flatten the

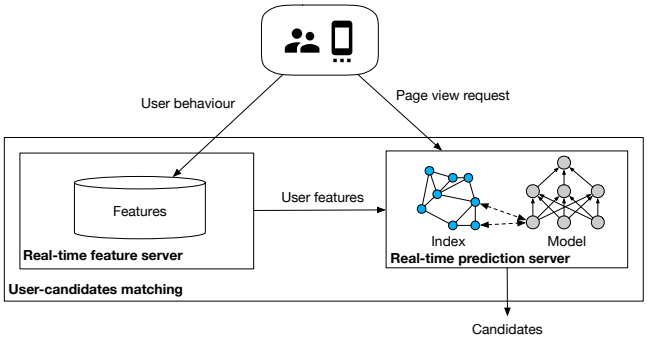


Figure 2: Online Serving System. The NANN service is provided by real-time prediction server, where graph-based index and deep neural network constitute a unified Tensorflow graph. The NANN service receives the user features from real-time feature server and output the retrieved candidate items to downstream task directly.

landscape of $s_u(\mathbf{e}_v)$ w.r.t. $\mathbf{e}_v$ via training on adversarially perturbed $\tilde{\mathbf{e}_v}$.

In our case, our solutions to maximize $s_u(\mathbf{e}_v)$ are limited to corpus $\mathcal{V}$. Hence, we mainly focus on the landscape of $s_u(.)$ around each $\mathbf{e}_v$ instead of the overall landscape. We formulate the training objective in terms of flatness as follow:

$$\begin{aligned} \mathcal{L}_{AUX} &= \sum_u \sum_{v \in \mathcal{Y}_u} s_u(\mathbf{e}_v) \log \frac{s_u(\mathbf{e}_v)}{s_u(\tilde{\mathbf{e}_v})} \\ \tilde{\mathbf{e}_v} &= \mathbf{e}_v + \Delta \end{aligned} \tag{4}$$

where $\mathcal{Y}_u$ consists of the labels from both true distribution and noise distribution for each $u \in \mathcal{U}$ according to NCE.

As described in Equation (4), the flatness is translated into a trainable objective: if $s_u(\mathbf{e}_v)$ has a flat landscape w.r.t $\mathbf{e}_v$, it should be robust, even invariant, to small perturbations around $\mathbf{e}_v$. Here, we use the Kullback–Leibler divergence to penalize the discrepancy between $s_u(\mathbf{e}_v)$ and $s_u(\tilde{\mathbf{e}_v})$ as $s_u(.)$ stands for the probability of user u engages with item v. As mentioned in Equation (3), the $\mathcal{L}_{AUX}$ combined with $\mathcal{L}_{NCE}$ constitute the final loss $\mathcal{L}_{all}$. The key to the auxiliary task is the direction and magnitude of perturbation Δ, where the adversarial gradient method comes into play.

In detail, we generate the adversarial examples by fast gradient sign method (FGSM) [11], which computes the perturbation as:

$$\Delta = \epsilon \text{sign}(\nabla_{\mathbf{e}_v} s_u(\mathbf{e}_v)) \tag{5}$$

where the $\nabla_{\mathbf{e}_v} s_u(\mathbf{e}_v)$ stands for the gradient of $s_u(\mathbf{e}_v)$ w.r.t. $\mathbf{e}_v$ that can be easily computed by backpropagation and the max-norm of perturbation Δ is bounded by ϵ.

Put simply, we achieve the search with the arbitrary measure without utilizing the convexity of $s_u(\mathbf{e}_v)$. Instead, our framework is built upon the flatness of $s_u(\mathbf{e}_v)$ w.r.t each $\mathbf{e}_v$, which can be achieved with a simple yet effective auxiliary task.

4 SYSTEM IMPLEMENTATION

Figure 2 illustrates the online serving architecture of the proposed method. The framework is flexible to use and maintain since we integrate graph-based index with the deep neural network and form a unified Tensorflow graph. The neural network inference and graph-based retrieval of NANN can thus serve as a unified module.

Here, we mainly emphasize the online serving efficiency optimizations of our proposed method, which are based on the Tensorflow framework.

4.1 Mark with Bitmap

To ensure the online retrieval performance, it is of importance to increase the outreach of candidate items within limited rounds of neighborhood propagation, as shown in Algorithm 2. Hence, we need to mark the visited items and bypass them to traverse further. The idea of Bitmap comes to mind as the v is serially numbered. We invent the Bitmap procedure by building C++ custom operators (Ops) within the Tensorflow framework. We summarize the performance of Bitmap Ops in terms of queries per second (QPS) and response time (RT) in milliseconds in Table 1.

Table 1: Different implementations of "Mark" procedure

$\{ef_2, ef_1, ef_0\}$	$\{T_2, T_1, T_0\}$	Ops	QPS	RT (ms)
$\{200, 500, 1000\}$	$\{1, 1, 2\}$	Raw	185	23.4
		Bitmap	**624**	**6.3**

1 GPU (Nvidia T4); 32 CPU cores (Intel(R) Xeon(R) Platinum 8163).
Deep neural network is accelerated by half-precision and XLA.

We test the performance of Bitmap Ops with the model architecture of $s_u(\mathbf{e}_v)$ deployed in production, of which detailed configuration will be introduced in Section 5. We traverse a three-layer graph-based index with the $|\mathcal{V}|$ equal to 1,300,000 and tune the parameters in Algorithm 2 to control the number of candidates, roughly 17,000 for the benchmark testing, to be evaluated. As demonstrated Table 1, our custom Bitmap Ops significantly outperform the Tensorflow Raw Set Ops.

4.2 Dynamic Shape with XLA

XLA (Accelerated Linear Algebra) is a domain-specific compiler for linear algebra that can accelerate the TensorFlow model [1]. XLA can automatically optimize the model execution in terms of speed and memory usage by fusing the individual Tensorflow Ops into coarsen-grained clusters. Our model has achieved a ~3x performance improvement with the help of XLA. However, it requires all tensors of the computation graph to have fixed shapes and compiled codes are specialized to concrete shapes. In our scenario, the number of unvisited items $|C|$ to be evaluated by $s_u(\mathbf{e}_v)$ is dynamic for each neighborhood propagation in Algorithm 2. Therefore, we present an "auto-padding" strategy to transform the dynamic shapes, e.g., $|C|$ in Algorithm 2, to certain predefined and fixed shapes. In detail, we set in advance a grid of potential shapes of $|C|$ and generate compiled codes for these predefined shapes with XLA's Just-in-Time (JIT) compilation, which is triggered by replaying the logs from the production environment. For online inference, the "auto-padding" strategy automatically pad the tensor with size $|C|$ to its nearest greater point on the grid and execute efficiently with its corresponding compiled code by XLA, and slice the tensor to its

[1] https://www.tensorflow.org/xla

original shape afterward. In short, we extend the capacity of XLA to dynamic shapes with an automatic "padding-slicing" strategy.

5 EXPERIMENTS

We study the performance of the proposed method as well as present the corresponding analysis in this section. Besides comparison to baseline, we put more emphasis on the retrieval performance of NANN and the corresponding ablation study due to the inadequacies of directly related works. Experiments on both an open-source benchmark dataset and an industry dataset from Taobao are conducted to demonstrate the effectiveness of the proposed method. We observe that our proposed method can significantly outperform the baseline and achieve almost the same retrieval performance as its brute-force counterpart with much fewer computations.

5.1 Setup

5.1.1 Datasets. We do experiments with two large-scale datasets: 1) a publicly accessible user-item behavior dataset from Taobao called UserBehavior [2]; 2) a real industry dataset of Taobao collected from traffic logs. Table 2 summarizes the main statistics for these two datasets.

Table 2: Statistics of evaluation datasets

	UserBehavior	Industrial Data of Taobao
# of users	976,779	100 million
# of items	4,163,442	1.3 million
# of records	85,384,110	375 million

UserBehavior. UserBehavior is a subset of Taobao user behaviors for recommendation problems with implicit feedback. Each record includes user ID, item ID, item category ID, behavior type, and timestamp. The behavior type indicates how the user interacts with the item, including click, purchase, adding items to the shopping cart, and adding items to favorites. We filter some of the users with high sparsity and keep the users with at least 10 behaviors. Suppose that the behaviors of user u be $(b_{u_1}, \ldots, b_{u_k}, \ldots, b_{u_n})$, the task is to predict $b_{u_{k+1}}$ based on the preceding behaviors. The validation and test sets are constituted by the samples from randomly selected 10,000 users respectively. We take the $\lceil l_u/2 \rceil$-th (l_u denotes the length of behavior sequence for user u) behavior of each u as ground truth and predict it based on all behaviors before.

Industrial Data of Taobao. The industry dataset is collected from the traffic logs in the Taobao platform, which is organized similarly to UserBehavior but with more features and records. The features of the industry dataset are mainly constituted by user profile, user behavior sequence, and item attributes. The validation and test sets are randomly draw from the traffic logs of the next day, which contains roughly 500,000 users.

5.1.2 Metrics. We use recall-all@M, recall-retrieval@M, recall-Δ@M, coverage@M to evaluate the effectiveness of our proposed method. In general, for a user u, the recall can be defined as

$$\text{recall}(\mathcal{P}_u, \mathcal{G}_u)@M(u) = \frac{|\mathcal{P}_u \cap \mathcal{G}_u|}{|\mathcal{G}_u|}.$$

[2] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

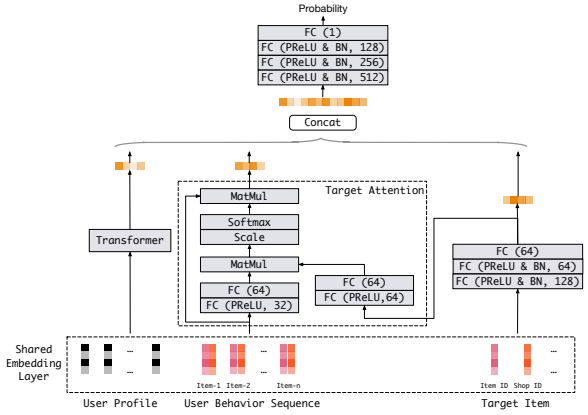


Figure 3: Detailed network architecture for the Industry and UserBehavior dataset.

where $\mathcal{P}_u(|\mathcal{P}_u| = M)$ denotes the set of retrieved items and $\mathcal{G}_u$ denotes the set of ground truths.

The capacity of a trained scoring model $s_u(\mathbf{e}_v)$ is assessed by exhaustively evaluating the corpus $\mathcal{V}$ for $u \in \mathcal{U}$, namely,

$$\text{recall-all}@M(u) = \text{recall}(\mathcal{B}_u, \mathcal{G}_u)@M(u)$$

where $\mathcal{B}_u = \text{argTopk}_{v \in \mathcal{V}}\, s_u(\mathbf{e}_v)$ ($|\mathcal{B}_u| = M$) is the set of precisely top-k scored items that can be produced by brute-force scanning.

Suppose that we traverse the graph-based index by $s_u(\mathbf{e}_v)$ and retrieve relevant items $\mathcal{R}_u$ ($|\mathcal{R}_u| = M$|) for each user u, the retrieval recall then can be evaluated by,

$$\text{recall-retrieval}@M(u) = \text{recall}(\mathcal{R}_u, \mathcal{G}_u)@M(u)$$

Correspondingly, the retrieval loss in terms of recall introduced by graph-based index can be defined as,

$$\text{recall-}\Delta@M(u) = \frac{\text{recall-all}@M - \text{recall-retrieval}@M}{\text{recall-all}@M}.$$

Furthermore, we make use of coverage@$M(u)$ to describe the discrepancy between the brute-force scanning and the retrieval. Formally,

$$\text{coverage}@M(u) = \frac{|\mathcal{R}_u \cap \mathcal{B}_u|}{|\mathcal{B}_u|}$$

From now on, we refer to the retrieval quality as the consistency between the items from retrieval and those from the brute-force, measured by recall-Δ@M and coverage@M.

Finally, we take the average over each u to obtain the final metrics, where u is from the testing set.

5.1.3 Model architecture. The model architecture (denoted as DNN w/ attention) is illustrated in Figure 1, which contains user network, target attention network, item network, and score network. All features are encoded after the shared embedding layer by mapping from categorical variables to dense vectors.

We demonstrate the detailed model architectures for both the industrial dataset and UserBehavior dataset in Figure 3. The deep models are built upon a variety of user and item features. These raw features are mapped to n-dimensional vectors with $n = 16$ by the embedding layer before feeding into the models. For the industrial dataset, the user profile features are encoded by the user network

with a transformer encoder [3] after passing the embedding layer. The item network is responsible for modeling the item features and composed of three fully connected layers. The relevance of the target item to the user behavior sequence is modeled by the target attention. The outputs of the user network, target attention, and item network are concatenated and fed into the scoring network to obtain the user-item preference score finally. For the UserBehavior dataset, it is slightly different from industrial dataset. As there is no user profile features but the user behavior sequence for each $u \in \mathcal{U}$, the user network is simply a sum-pooling layer over the user behavior sequence.

To measure the model capacity and retrieval performance of different model structures, we also conduct experiments on the following model structures: 1) DNN w/o attention, which replaces the target attention network with a simple sum-pooling over the embeddings of user behavior sequence; 2) two-sided, which only consists of user embedding (the concatenation of the output of user network and the sum-pooling over the embeddings of user behavior sequence) and item embedding, and calculate the user-item preference score by inner product.

5.1.4 Implementation details. Our large-scale model training is conducted on the distributed training system XDL [35]. Given the dataset and model structure, we train the model with the loss function defined in Equation (3). Adam optimizer with learning rate 3e-3 is adopted to minimize the loss. The ϵ of FGSM is set to 1e-2 for the industry dataset and 3e-4 for the UserBehavior dataset. We optimize the models by NCE and assign each label from the true distribution with 19 and 199 labels from noise distribution for the industry dataset and UserBehavior dataset respectively.

After training, we extract the item feature after the item network for all valid items to build the HNSW graph. The standard index build algorithm [21] is used, the number of established connections is set to 32, size of the dynamic candidate list in the graph construction stage is set to 40.

In the retrieval stage, we exhaustively calculate the scores of items in layer 2, which consists of millesimal items of entire vocabulary and can be scored in one batch efficiently. Then top-k relevant items, with k=ef_2, are retrieved as enter points for the following retrieval. The default retrieval parameter is set as $\{ef_2, ef_1, ef_0\} = \{100, 200, 400\}$, $\{T_2, T_1, T_0\} = \{1, 1, 3\}$, described in Algorithms 1 and 2. Without further claim, we report top-200 ($M = 200$) metrics for final retrieved items.

All the hyper-parameter are determined by cross-validation.

5.2 Results

5.2.1 Comparison to Baselines. We compare with the baseline method SL2G, which directly leverages HNSW-retrieval with the deep model in the HNSW graph constructed by $l2$ distance among $\mathbf{e}_v$. The comparison results of different methods are shown in Figure 4. Each x-axis stands for the ratio of the number of traversed items to $|\mathcal{V}|$ for reaching the final candidates.

First of all, NANN achieves great improvements on recall and coverage in comparison with SL2G across different numbers of traversed items for the two datasets. Especially, NANN outperforms SL2G by a larger margin when we evaluate a smaller portion of items to reach the final items.

Second, NANN performs on par with its brute-force counterpart by much fewer computations. Especially, NANN hardly plagues retrieval quality and achieves 0.60% recall-Δ and 99.0% coverage with default retrieval parameter when applied to industrial data of Taobao. Moreover, the model capacity and robustness indicated by recall-all can also benefit from the defense against moderate adversarial attacks.

Finally, NANN can rapidly converge, in terms of traversed items, to a promising retrieval performance. As described by the curvatures of Figure 4, only 1% ~2% of $\mathcal{V}$ need to be evaluated to reach a satisfying retrieval quality for the two datasets.

5.2.2 Beam-retrieval vs HNSW-retrieval. Figure 4 demonstrates the recall and coverage for the Beam-retrieval (the "NANN" curve) and the original HNSW-retrieval (the "NANN-HNSW" curve) respectively. As shown in these figures, Algorithm 2 outperforms the HNSW-retrieval version in two ways: 1) it performs consistently better across different numbers of traversed items; 2) it converges to the promising retrieval quality more rapidly. Moreover, as shown in Figure 6, the while-loop of HNSW-retrieval results in redundant rounds of neighborhood propagation in the ground layer which is unnecessary for recall and coverage.

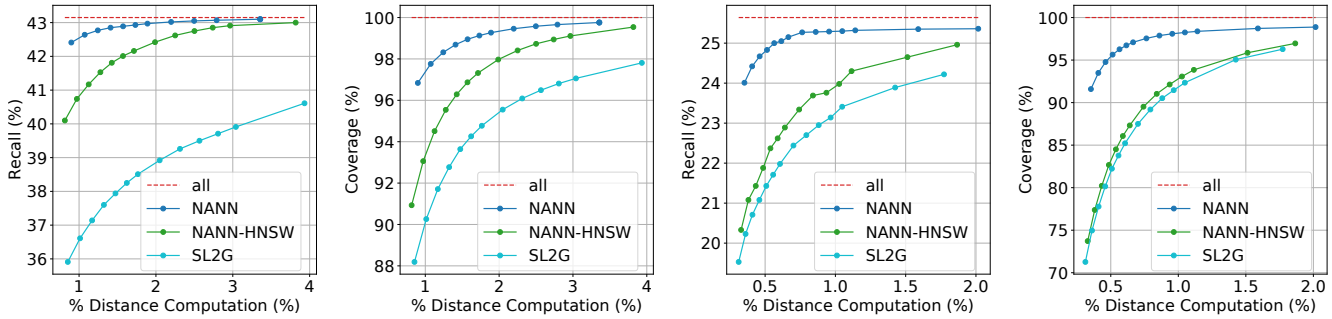
Table 3: Results of different model architectures on industry and UserBehavior dataset.

Dataset	Aux loss	model	recall-retrieval	recall-all	recall-Δ
Industry	w/o	two-sided	28.8%	28.9%	0.35%
		DNN w/o attention	33.9%	34.1%	0.59%
		DNN w/ attention	39.2%	42.8%	8.55%
	w/	two-sided	30.1%	30.2%	**0.33%**
		DNN w/o attention	34.1%	34.2%	**0.29%**
		DNN w/ attention	**42.9%**	**43.2%**	**0.60%**
User Behavior	w/o	two-sided	11.3%	11.4 %	0.74%
		DNN w/o attention.	12.8%	13.1%	2.30%
		DNN w/ attention	23.9%	24.7%	3.48%
	w/	two-sided	12.4%	12.5%	**0.40%**
		DNN w/o attention	13.1%	13.3%	1.20%
		DNN w/attention	**24.9%**	**25.6%**	3.00%

5.2.3 Effectiveness of adversarial gradient training. In Figure 5, we traverse the similarity graph with Beam-retrieval and demonstrate the effectiveness of the defense against adversarial attacks. We observe that NANN is constantly superior to the model without adversarial training across the different degrees of traversal.

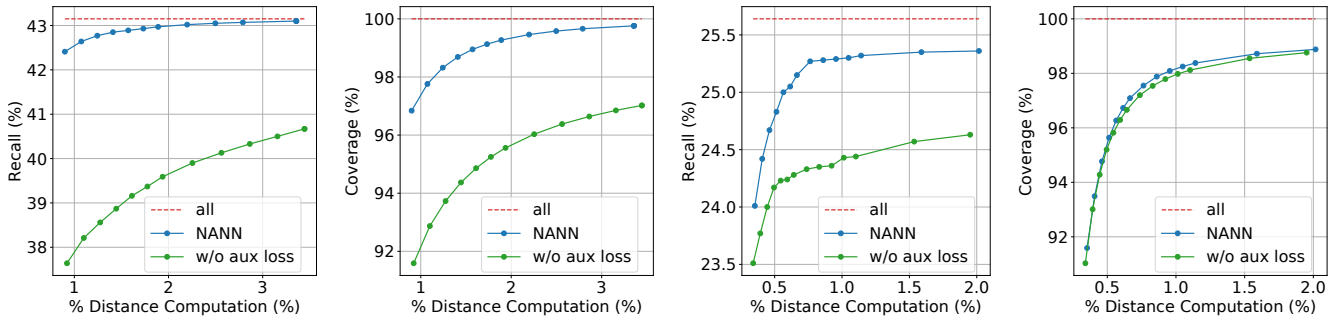
We also investigate the effects of FGSM on different model architectures. As indicated by recall-all and recall-Δ in Table 3, we empirically show that more complex models usually generalize better and achieve higher performances but may deteriorate the retrieval quality. Based on this observation, we claim that the growing discrepancy between recall-all and recall-retrieval may stem from the higher heterogeneity between similarity measures, and thus exploit adversarial training to mitigate the discrepancy. The default

[3] https://www.tensorflow.org/tutorials/text/transformer



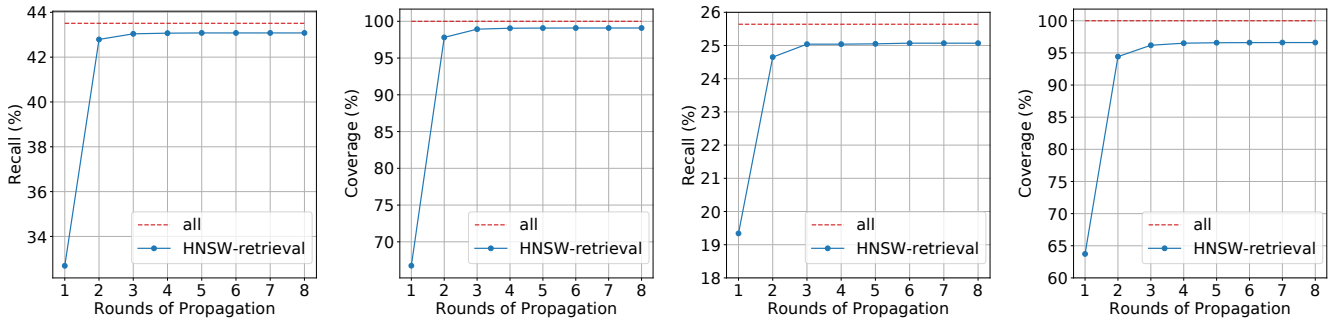
(a) Recall-Industry (b) Coverage-Industry (c) Recall-UserBehavior (d) Coverage-UserBehavior

Figure 4: Results of our proposed NANN and SL2G on Industry and UserBehavior dataset.



(a) Recall-Industry (b) Coverage-Industry (c) Recall-UserBehavior (d) Coverage-UserBehavior

Figure 5: The effect of adversarial gradient training on Industry and UserBehavior dataset.



(a) Recall-Industry (b) Coverage-Industry (c) Recall-UserBehavior (d) Coverage-UserBehavior

Figure 6: Neighborhood propagation in ground layer

retrieval parameter is used for the comparison. As shown in Table 3, the performance of all model architectures ranging from simple to complex can benefit from the adversarial training; FGSM can greatly improve the retrieval quality, especially for more complex models.

5.2.4 Analysis for adversarial gradient training. Figure 7 shows the reaction of model to adversarial attack after model training. We define the Δ of the adversarial attack as $\epsilon \cdot \text{rand}(-1, 1)$ akin to FGSM and compare the robustness of different models by visualizing $|s_u(\mathbf{e}_v) - s_u(\mathbf{e}_v + \Delta)|$. Figure 7 is the histogram of $|s_u(\mathbf{e}_v) - s_u(\mathbf{e}_v + \Delta)|$ where $v \in \text{argTopk}_{v \in \mathcal{V}} s_u(\mathbf{e}_v)$. As demonstrated in Figure 7, the retrieval quality empirically correlates to the robustness of model when faced with adversarial attack: 1) the greater right-skewed distribution of $|s_u(\mathbf{e}_v) - s_u(\mathbf{e}_v + \Delta)|$ for model without attention demonstrates its superior robustness to model with attention, which is consistent with their recall-Δ in Table 3; 2) the retrieval quality of model with attention can be significantly improved by FGSM,

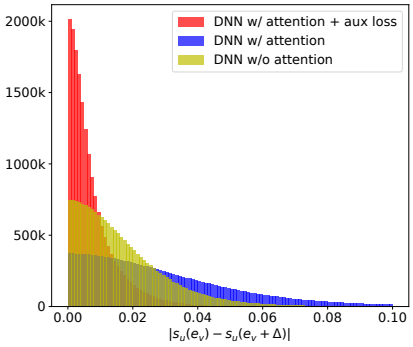
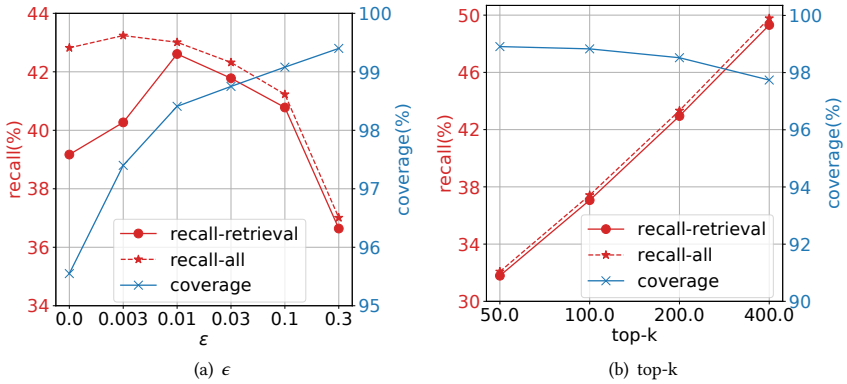


Figure 7: Reaction to small perturbations on the Industry dataset.



(a) ϵ (b) top-k

Figure 8: Sensitivity Analysis for ϵ in FGSM and top-k of ground layer in Beam-retrieval on the Industry dataset.

and meanwhile its distribution of $|s_u(\mathbf{e}_v) - s_u(\mathbf{e}_v + \Delta)|$ becomes more skewed to the right with adversarial training.

5.2.5 *Sensitivity analysis.*

Magnitude of ϵ. Figure 8(a) shows the correlation between ϵ and retrieval quality measured by coverage. In general, the retrieval quality is positively correlated with the magnitude of ϵ. Besides, adversarial attacks can be beneficial to the overall performance measured by recall-all with a mild magnitude of ϵ, but harmful when ϵ gets excessively large. Hence, the magnitude of ϵ plays an important role in the balance between retrieval quality and overall performance.

Different top-k. Figure 8(b) shows the effects of our proposed method on different k for the final top-k retrieved items in Algorithm 1. NANN performs consistently well across different k. The retrieval quality can be still guaranteed despite retrieving with larger k. Therefore, our method is insensitive to k in general.

5.3 Online Results

Our proposed method is evaluated with real traffic in the Taobao display advertising platform. The online A/B experiments are conducted on main commercial pages within Taobao App, such as the "Guess What You Like" page, and last more than one month. In general, these pages deal with more than one hundred thousand page view requests per second and provides services for at least one billion users. Our online A/B experiments are conducted on randomly selected 5% of the overall page view requests. The online baseline is the latest TDM method with Bayes optimality under beam search [37–39]. For a fair comparison, we only substitute TDM, one of the channels in the candidate generation stage, with NANN and maintain other factors like the number of candidate items that delivered to the ranking stage unchanged. Two common metrics for online advertising are adopted to measure online performance: Click-through Rate (CTR) and Revenue per Mille (RPM).

$$\text{CTR} = \frac{\text{\# of clicks}}{\text{\# of impressions}}, \quad \text{RPM} = \frac{\text{Ad revenue}}{\text{\# of impressions}} \times 1000.$$

NANN significantly contributes up to 2.4% CTR and 3.1% RPM promotion compared with TDM, which demonstrates the effectiveness of our method in both user experience and business benefit.

Moreover, the efficient implementation of NANN introduced in Section 4 facilitates us to benefit from NANN without sacrificing the RT and QPS of online inference. In production, NANN meets the performance benchmark displayed in Table 1. Now, NANN has been fully deployed and provides the online retrieval service entirely in the Taobao display advertising platform, e.g. the online serving infrastructure of ADIN [17] has been upgraded with NANN recently, which results in noticeable online improvements.

6 CONCLUSION

In this paper, we propose a lightweight approach to integrating post-training graph-based index with the arbitrarily advanced model. We present both heuristic and learning-based methods to ensure the retrieval quality: 1) our proposed Beam-retrieval can significantly outperform the existing search on graph method under the same amount of computation; 2) we inventively introduce adversarial attack into large-scale retrieval problems to benefit both the retrieval quality and model robustness. Extensive experimental results have already validated the effectiveness of our proposed method. In addition, we summarize in detail the hands-on practices of deploying NANN in Taobao display advertising where NANN has already brought considerable improvements in user experience and commercial revenues. We hope that our work can be broadly applicable to domains beyond recommender system such as web search and content-based image retrieval. In the future, we hope to further uncover the underlying mechanisms that govern the applicability of adversarial attacks to large-scale retrieval problems.

7 ACKNOWLEDGMENTS

We deeply appreciate Xiaoqiang Zhu, Jingwei Zhuo, Wei Dai and Xiang Li for their helpful suggestions and discussions. Thank Peng Sun, Yuxin Liu, Jingshan Lv, Huimin Yi, Yuanxing Zhang and Kaixu Ren for implementing the key components of the serving infrastructure and training infrastructure. Thank Haiping Huang and Linhao Wang for necessary engineering supports.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Franz Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)* 23, 3 (1991), 345–405.

[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*. 257–264.

[4] Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning*. 112–119.

[5] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. 2019. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment* 2019, 12 (2019), 124018.

[6] Ryan R Curtin and Parikshit Ram. 2014. Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7, 4 (2014), 229–253.

[7] Ryan R Curtin, Parikshit Ram, and Alexander G Gray. 2013. Fast exact max-kernel search. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 1–9.

[8] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. 2015. Natural neural networks. *Advances in neural information processing systems* 28 (2015).

[9] Steven Fortune. 1995. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean geometry* (1995), 225–265.

[10] Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. 2020. Deep Retrieval: Learning A Retrievable Structure for Large-Scale Recommendations. *arXiv preprint arXiv:2007.07203* (2020).

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[12] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 297–304.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Flat minima. *Neural computation* 9, 1 (1997), 1–42.

[15] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.

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

[17] Yuchen Jiang, Qi Li, Han Zhu, Jinbei Yu, Jin Li, Ziru Xu, Huihui Dong, and Bo Zheng. 2022. Adaptive Domain Interest Network for Multi-domain Recommendation. *arXiv preprint arXiv:2206.09672* (2022).

[18] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-interest network with dynamic routing for recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2615–2623.

[19] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2017. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913* (2017).

[20] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[21] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[22] Stanislav Morozov and Artem Babenko. 2019. Relevance Proximity Graphs for Fast Relevance Retrieval. *arXiv preprint arXiv:1908.06887* (2019).

[23] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11, 1 (2002), 28–46.

[24] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2685–2692.

[25] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 931–939.

[26] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free! *arXiv preprint arXiv:1904.12843* (2019).

[27] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *arXiv preprint arXiv:1405.5869* (2014).

[28] Anshumali Shrivastava and Ping Li. 2015. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th international conference on world wide web*. 981–991.

[29] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. 2017. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2107–2116.

[30] Ayan Sinha, Zhao Chen, Vijay Badrinarayanan, and Andrew Rabinovich. 2018. Gradient adversarial training of neural networks. *arXiv preprint arXiv:1806.08028* (2018).

[31] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast item ranking under neural network based measures. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 591–599.

[32] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quanquan Gu. 2021. On the convergence and robustness of adversarial training. *arXiv preprint arXiv:2112.08304* (2021).

[33] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. 2018. Hessian-based analysis of large batch training and robustness to adversaries. *Advances in Neural Information Processing Systems* 31 (2018).

[34] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems* 30, 9 (2019), 2805–2824.

[35] Yuanxing Zhang, Langshi Chen, Siran Yang, Man Yuan, Huimin Yi, et al. 2022. PICASSO: Unleashing the Potential of GPU-centric Training for Wide-and-deep Recommender Systems. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE.

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

[37] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. Joint optimization of tree-based index and deep model for recommender systems. *Advances in Neural Information Processing Systems* 32 (2019).

[38] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1079–1088.

[39] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning optimal tree models under beam search. In *International Conference on Machine Learning*. PMLR, 11650–11659.