

An Efficient Embedding Based Ad Retrieval with GPU-Powered Feature Interaction

Yifan Lei*

Jiahua Luo*†

{yifanlei,kennyjhluo}@tencent.com
Tencent Inc.
Shenzhen, Guangdong, China

Lifeng Wang

fandywang@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Haijie Gu

jerrickgu@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Tingyu Jiang

travisjiang@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Bo Zhang

peanutzhang@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Dapeng Liu

rocliu@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Huan Yu

huanyu@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Zhaoren Wu

jasonzrwu@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Jie Jiang

zeus@tencent.com

Tencent Inc.

Shenzhen, Guangdong, China

Abstract

In large-scale advertising recommendation systems, retrieval serves as a critical component, aiming to efficiently select a subset of candidate ads relevant to user behaviors from a massive ad inventory for subsequent ranking and recommendation. The Embedding-Based Retrieval (EBR) methods modeled by the dual-tower network are widely used in the industry to maintain both retrieval efficiency and accuracy. However, the dual-tower model has significant limitations: the embeddings of users and ads interact only at the final inner product computation, resulting in insufficient feature interaction capabilities. Although DNN-based models with both user and ad as input features, allowing for early-stage interaction between these features, are introduced in the ranking stage to mitigate this issue, they are computationally infeasible for the retrieval stage.

To bridge this gap, this paper proposes an efficient GPU-based feature interaction for the dual-tower network to significantly improve retrieval accuracy while substantially reducing computational costs. Specifically, we introduce a novel compressed inverted list designed for GPU acceleration, enabling efficient feature interaction computation at scale. In other words, we implemented the Wide & Deep model architecture in the retrieval stage. To the best of our knowledge, this is the first framework in the industry to successfully implement Wide & Deep in a retrieval system. We apply this model to the real-world business scenarios in Tencent

Advertising, and experimental results demonstrate that our method outperforms existing approaches in offline evaluation and has been successfully deployed to Tencent's advertising recommendation system, delivering significant online performance gains. This improvement not only validates the effectiveness of the proposed method, but also provides new practical guidance for optimizing large-scale ad retrieval systems.

CCS Concepts

• Information System → Retrieval efficiency and scalability.

Keywords

Embedding-Based Retrieval, Implicit Feature Interaction, Explicit Feature Interaction, GPU, Dual-Tower Network

ACM Reference Format:

Yifan Lei, Jiahua Luo, Tingyu Jiang, Bo Zhang, Lifeng Wang, Dapeng Liu, Zhaoren Wu, Haijie Gu, Huan Yu, and Jie Jiang. 2025. An Efficient Embedding Based Ad Retrieval with GPU-Powered Feature Interaction. In *Proceedings of XXXXX (Conference'17)*. ACM, New York, NY, USA, 9 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Online advertising recommendation systems are essential tools that leverage user data to deliver personalized ad content, enhancing user engagement and maximizing advertiser returns. These systems have become crucial in powering large-scale online advertising platforms, driving business growth while ensuring tailored user experiences.

A typical modern advertising recommendation system usually has a multi-stage funnel architecture: retrieval, pre-ranking, and ranking stages [9, 40, 39]. As the first stage, retrieval rapidly scans through millions of candidate ads to generate a manageable subset of high-potential ads. This stage relies on extremely efficient models to deal with the whole advertising database. A prevalent approach is the dual-tower architecture, where user and ad features

*Both authors contributed equally to this research.

†Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, Washington, DC, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06

<https://doi.org/XXXXXXX.XXXXXXX>

are independently encoded via deep neural networks (DNNs) to generate embeddings, followed by an inner product operation to compute relevance scores.

For instance, in Tencent's advertising system, retrieval rapidly scans millions of ads to generate a candidate subset of ads using a dual-tower Embedding-Based Retrieval (EBR) [18, 27] with an Approximate Nearest Neighbor (ANN) search library. In the pre-ranking and ranking stages, the selected candidate ads are further expanded into respective creative formats and evaluated using more compute-intensive models to select the optimal candidate ads for user display. This hierarchical funnel framework is illustrated in Figure 1.

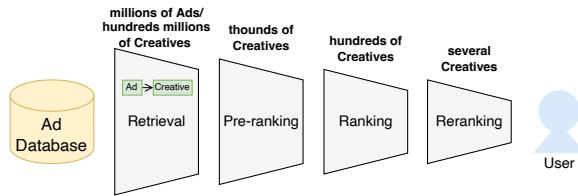


Figure 1: The overview of the multi-stage funnel architecture for industrial advertising recommendation systems

The dual-tower model, however, cannot model the interaction between the embeddings of users and ads until the final inner product computation, which leads to insufficient feature interaction capabilities [17]. Although advanced models such as Inner Product Neural Network (IPNN), Wide & Deep, DCN and Cold [33, 14, 8, 25, 39] have been developed for the pre-ranking and ranking stages to capture complex feature interactions, their substantial computational cost makes them infeasible for the efficiency constraints of retrieval tasks. Recently, many DNN-based retrieval models have been proposed [42, 13, 7]. As they use a DNN scorer for retrieval, these models, however, still suffer from significant query time increases during the retrieval stage, which may further reduce the time available for pre-ranking or ranking, potentially diminishing the overall system's effectiveness.

To address these issues, we propose a retrieval model integrating both implicit and explicit feature interactions through a novel GPU-accelerated operator. Specifically, we investigate the IPNN [33, 14] and Wide & Deep [8] models to the retrieval stage. While the IPNN emphasizes implicit feature interactions, the Wide & Deep model specializes in capturing explicit interactions, making their combination well-suited for joint modeling of nuanced user-ad relationships. Despite their wide usage in later stages of the funnel in pre-ranking and ranking models, these two models have rarely been applied in retrieval due to their computational expense.

To integrate these feature interaction models into the retrieval stage and efficiently handle millions of ads during the retrieval stage, we introduce two strategies for implicit and explicit feature interaction. For implicit feature interaction, we adopt the dimension extension method to concatenate the implicit feature expression into the original user and item embedding, making it still a dual-tower model. For explicit feature interaction, we propose a novel compressed inverted list designed for GPU acceleration. Unlike conventional sparse matrix operations, our method makes use of

the binary nature of the feature interaction matrix and organizes ad-feature interactions into GPU-friendly inverted lists. Although this introduces initial pre-processing overhead, since the index remains constant for a period of time, the cost can be amortized across repeated queries. With two key considerations, minimizing data transferring between devices and ensuring load balance, our approach organizes data into compact, query-optimized structures, significantly reducing query latency by up to 7x compared to sparse matrix multiplication methods such as the cuSPARSE [31] library, a GPU implementation of the sparse matrix multiplication provided by Nvidia. In summary, the contributions of our paper are as follows:

- We integrate both implicit and explicit feature interactions into an industry-scale retrieval model. With consideration of the computational overhead of each component in the model, our design balances computational efficiency and model expressiveness, enabling further optimization of complex feature interactions operators.
- Combining the characteristics of operators, we propose a GPU-powered inverted list index, which significantly accelerates the computation of explicit feature interaction, making its application feasible in retrieval scenarios.
- Our extensive experiments on real-world commercial datasets show that the proposed model does improve the revenue of the ad recommendation, and the proposed GPU-powered inverted list index can outperform the baseline GPU sparse matrix multiplication method by a large margin.

2 Related Work

Dual-tower Network In advertising recommendation systems, retrieval stage aims to retrieve top k relevant ads from a large pool of candidates for the subsequent ranking, requiring models that are computationally lightweight, efficient, high-throughput, and low-latency. A prevalent approach in industry is the dual-tower architecture, where user and advertisement features are independently encoded via deep neural networks (DNNs) to generate embeddings, followed by an inner product operation to compute relevance scores. Commonly used dual-tower models include DSSM and its variants (e.g., CDSSM[36], MV-DSSM[12]). DSSM employs deep neural networks to map query and document features into a shared semantic space for relevance matching. CDSSM extends DSSM by incorporating convolutional neural networks to capture local n-gram patterns in text for enhanced semantic representation. MV-DSSM extends the DSSM architecture by incorporating multiple views of data (e.g., text, images, and metadata) into separate towers, enabling joint learning of heterogeneous features for improved semantic matching. YouTube DNN [9] is designed for recommendation systems, where one tower encodes user features and the other encodes item IDs using a softmax activation function, optimized for large-scale candidate retrieval. SARec[22] Integrates self-attention mechanisms into the dual-tower framework to model sequential user behavior, capturing long-range dependencies and temporal patterns for more accurate recommendations. BERT4Rec[37] is a transformer-based sequential recommendation model that leverages bidirectional self-attention and masked language model pre-training to capture contextual information in user behavior sequences, enhancing recommendation performance. The dual-tower model retrieve the top ads

by maximizing the relevance scores to the user, which is known as Nearest Neighbor Search problem. To address the Nearest Neighbor Search problem efficiently, Approximate Nearest Neighbor (ANN) techniques are widely adopted, including Locality-Sensitive Hashing (LSH) [16, 23], Inverted File Index (IVF) [20], Hierarchical Navigable Small World (HNSW) [29], and Product Quantization (PQ) [19, 15], which trade a slight reduction in precision for significant efficiency gains. Clustering-based methods optimize retrieval by narrowing the search space, while graph-based approaches like HNSW use graph traversal for efficient high-dimensional retrieval. Quantization techniques like PQ reduce storage and computational overhead by compressing vector representations. ANN methods, particularly HNSW and IVF-PQ, are standard for large-scale vector retrieval due to their scalability and efficiency.

Feature Interaction In advertising recommendation systems, feature interaction algorithms are crucial for capturing feature relationships and enhancing model performance, particularly during the fine-ranking stage with small candidate sets, enabling precise ad ranking and scoring. The following are key algorithms in this domain: Factorization Machines (FM) [34] model second-order feature interactions via latent vector inner products, excelling in sparse data and widely used in recommendations and CTR prediction. Field-aware Factorization Machines (FFM) [21] extend FM by learning field-specific latent vectors, improving multi-field feature interaction modeling. Field-weighted Factorization Machines (FwFM) [32] introduces field interaction weights, learning the adaptive interaction weight among feature fields. Wide & Deep integrates explicit feature interactions with deep neural networks, designed to achieve a balance between memorization and generalization capabilities. Deep Cross Network (DCN) [38] combines deep networks with explicit interaction layers, learning high-order interactions while preserving low-order features. DeepFM integrates FM's second-order interactions with deep networks for end-to-end training, balancing expressiveness and efficiency. Inner Product-based Neural Network (IPNN) explicitly models interactions via inner products, capturing nonlinear feature combinations. Co-Action Network (CAN) [4] emphasizes feature pair synergies, flexibly modeling complex relationships in high-dimensional sparse data.

Inverted List The inverted list data structure is widely used in applications such as large-scale information retrieval and recommendation systems [10, 2]. To reduce storage overhead and enhance query performance, numerous optimization techniques have been developed for inverted list indices. For instance, Variable Byte (VB) and its variants [10, 11] employ a null suppression schema to compress the posting list. VB [10] leverages 7 bits in a byte to represent the data, with 1 bit indicating whether the current bit is the end of the current integer. Group VB [11] compresses 4 bytes at the same time and uses 2 bits to indicate the number of valid bytes per integer. PforDelta [43] proposes to compress the delta code of each list. The delta code of each integer is partitioned into blocks, and in each block, the smallest possible bit length b is chosen to represent the values in the block in b -bits. To improve the query performance of inverted lists, SIMDpforDelta [24], as a SIMD version of PforDelta, utilizes SIMD instructions to accelerate the processing of the inverted list index. In recent years, GPU-accelerated inverted list indices have been proposed to harness the parallel processing capabilities of GPUs [41]. For instance, Zhou et al. [41] demonstrated the

use of GPU-accelerated inverted lists for similarity search, enabling efficient parallel processing of batched queries. In [35], Shanbhag et al. proposed to use a cascading decompression schema optimized for GPU architecture to compress the inverted list.

3 Methodology

In this section, we introduce the proposed model in detail. First, we introduce the overview of the dual-tower network in the learning to rank framework. Then, we show the proposed implicit and explicit feature interaction modules for the dual-tower model. Finally, we present our proposed GPU inverted list and how it can accelerate the computation of the explicit feature interaction operator.

3.1 Learning-to-Rank based Dual-Tower Model in Ad Retrieval

From a mathematical perspective, retrieval can be defined as a preference function that selects a subset of high-scoring items from a candidate set. The scoring function is usually trained using the dual-network, where user and ad features are separately transformed to a low dimensional representation $\mathbf{h}_u \in \mathbb{R}^d$ and $\mathbf{h}_a \in \mathbb{R}^d$ respectively. For a dual-tower model, the preference score $s = s(u, a)$ of user u for ad a is computed as the inner product of the user representation \mathbf{h}_u and the ad representation \mathbf{h}_a

$$s(u, a) = \langle \mathbf{h}_u, \mathbf{h}_a \rangle = \mathbf{h}_u^\top \mathbf{h}_a \quad (1)$$

We utilize the LambdaRank framework[6] with pairwise logistic loss [28] and non-differentiable measurement metrics to train the model on a dataset where the labels represent the desired ranking outcomes. Specifically, the loss function for the ranking function contains two parts, pairwise logistic loss and measurement metrics, as defined as:

$$\mathcal{L} = \sum_{(i,j):a_i < a_j} \log(1 + e^{-(s_i - s_j)}) \cdot |\Delta NDCG_{ij}| \quad (2)$$

The metric NDCG is concerned with the ranking of top candidates, which matches the goal of retrieval—to return a top-N set from millions of candidates. It is formally defined as follows,

$$DCG_i = \sum_{i=1}^{i=D} \frac{2^{p_i} - 1}{\log(i+1)} \quad (3)$$

$$= \sum_{i=1}^{i=D} \frac{2^{D-i} - 1}{\log(i+1)} \quad (4)$$

$$NDCG_i = \frac{DCG_i}{maxDCG} \quad (5)$$

where D is the length of the sampled advertisement sequence. However, the LambdaRank algorithm, with its core metric $\Delta NDCG$, is originally designed for relevance ranking in document retrieval. It computes ranking cost solely from the relative order of items, ignoring their intrinsic value. This poses a limitation in ad ranking, where eCPM directly determines revenue in ad bidding like oCPM: misordering ads with divergent eCPMs results in greater financial loss than with similar eCPMs. To integrate this business metric, we refine the original LambdaRank loss.

$$\mathcal{L} = \sum_{(i,j):a_i < a_j} \log(1 + e^{-(s_i - s_j)}) \cdot \{|\Delta NDCG_{ij}| \odot |\Delta Value_{ij}| \} \quad (6)$$

where \odot denotes multiplication or addition operator. $\Delta Value$ can be defined as the loss of the physical value for the ground truth incurred by swapping the positions of ads a_i and a_j . Its physical implication is that misranking two ads with a significant commercial value difference leads to a greater penalty. In practical application, researchers or developers can select the most appropriate non-differentiable loss function in the LambdaRank framework according to specific scenario and data characteristics[6].

As the dual-tower model is efficient for retrieval tasks but struggles with complex feature interactions due to its separate architecture of user and ad features. To improve this, two new modules, implicit and explicit feature interaction, are introduced to the dual-tower in the following subsections. The implicit module uses latent space transformations to capture subtle dependencies, while the explicit module models higher-order feature interactions directly. These modules, when integrated into the dual-tower architecture, enhance its expressiveness without sacrificing scalability. The structures of the proposed modules are illustrated in Figure 2.

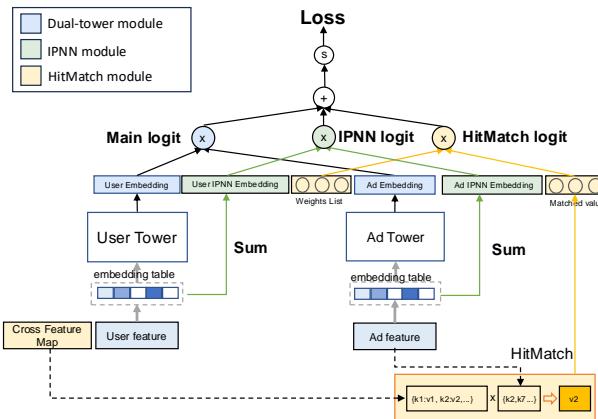


Figure 2: The proposed model architecture with three core components: a dual-tower network, an IPNN module, and a HitMatch module, which generate the main logit, IPNN logit, and HitMatch logit. These three logits are aggregated through summation at the final layer of the model.

3.2 Implicit Feature Interaction

In this subsection, we introduce the IPNN operator designed to capture the interaction between user and ad features. Let $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{v} = (v_1, v_2, \dots, v_m)$ represent the concatenation of a certain user and ad features, respectively, after being transformed through embedding tables and pooling operations. The IPNN operator is decomposed as

$$\sum \langle u_i, v_j \rangle = \langle \tilde{\mathbf{u}}, \tilde{\mathbf{v}} \rangle; \quad \tilde{\mathbf{u}} = \mathbf{W}^{(u)} \mathbf{u}; \quad \tilde{\mathbf{v}} = \mathbf{W}^{(v)} \mathbf{v}, \quad (7)$$

where $\mathbf{W}^{(u)} \in \mathbb{R}^{d \times n}$ and $\mathbf{W}^{(v)} \in \mathbb{R}^{d \times m}$ denote the weight matrices corresponding to the fields u and v , respectively. Therefore, by applying the IPNN operator to the dual-tower model, we can first compute $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ separately on the user and ad, respectively, and then perform the inner product operation at the top layer of the

model. In practical applications, by expanding the lengths of the user and ad vectors, we concatenate the summed $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ after the \mathbf{h}_u and \mathbf{h}_a respectively. This model remains as a dual-tower structure, maintaining high computational efficiency. Specifically, the new logit calculation formula is:

$$s(u, a) = \tilde{\mathbf{h}}_u \tilde{\mathbf{h}}_a = [\mathbf{h}_u, \tilde{\mathbf{u}}]^T [\mathbf{h}_a, \tilde{\mathbf{v}}], \quad (8)$$

where $[., .]$ denotes the vector concatenation.

3.3 Explicit Feature Interaction

The integration of the IPNN operator can improve the feature interaction capabilities for the dual-tower architecture to some extent. However, its structure remains inherently a dual-tower network which still exhibits limitations in feature interaction. In this subsection, we investigate a lightweight, non-dual-tower architecture, inspired from the widely-adopted Wide & Deep model in the ranking phase. Here we adapt it to the retrieval phase and name it HitMatch module throughout this paper. The proposed module is depicted in Figure 2, highlighted with yellow color. To minimize computational overhead, the wide part features are selected cautiously, only including their side information and the IDs of ads that users have historically interacted with. The side information includes multi-level ad categories, tags, advertiser IDs, and item IDs, etc. These historical behavior features are grouped as multiple sequences associated with the user ID. During the training process, the ad component in the side information feature looks up the matched feature values from the user sequences. It then performs a weighted sum using the associated learnable weights to generate the logit for the cross part.

Suppose that there are N ads and the wide component contains M cross features, where the associated weights are $\mathbf{w} = \{w_1, w_2, \dots, w_M\}$ and their values are $\mathbf{x} = \{x_1, x_2, \dots, x_M\}$. We denote whether the feature values empty or not by $\mathbf{L} \in \{0, 1\}^{N \times M}$, where each element $L_{a,i} = 1$ when ad a whose feature- i not empty, and otherwise $L_{a,i} = 0$. \mathbf{L} is highly sparse for a specific ad. The final score of ad a is calculated as

$$s(u, a) = \langle \tilde{\mathbf{h}}_u, \tilde{\mathbf{h}}_a \rangle + \sum_{i=1}^M w_i x_i L_{a,i} \quad (9)$$

Here, $\tilde{\mathbf{h}}_u$ and $\tilde{\mathbf{h}}_a$ correspond to the concatenated outputs of the user tower and the ad tower, respectively, derived from both the dual-tower structure and the IPNN module. In the inference phase, the inner product computation in the dual-tower part is very lightweight. However, the HitMatch operator is computationally expensive due to the large number of candidate ads, which can reach millions. Therefore, it is necessary to carefully design an efficient indexing operator to quickly compute results within a short time frame. In the next section, we will introduce the efficient indexing HitMatch operator that we have developed.

3.4 GPU Inverted List based HitMatch Operator

As shown in Equation 9, the HitMatch score can be calculated using a sparse matrix multiplication $\mathbf{L}\tilde{\mathbf{w}}$, where $\tilde{\mathbf{w}}_i = w_i x_i, \forall i$. Computing this sparse matrix multiplication, however, is still much slower than computing the dual-tower scores with a dense matrix multiplication. This significant performance gap renders a naive application

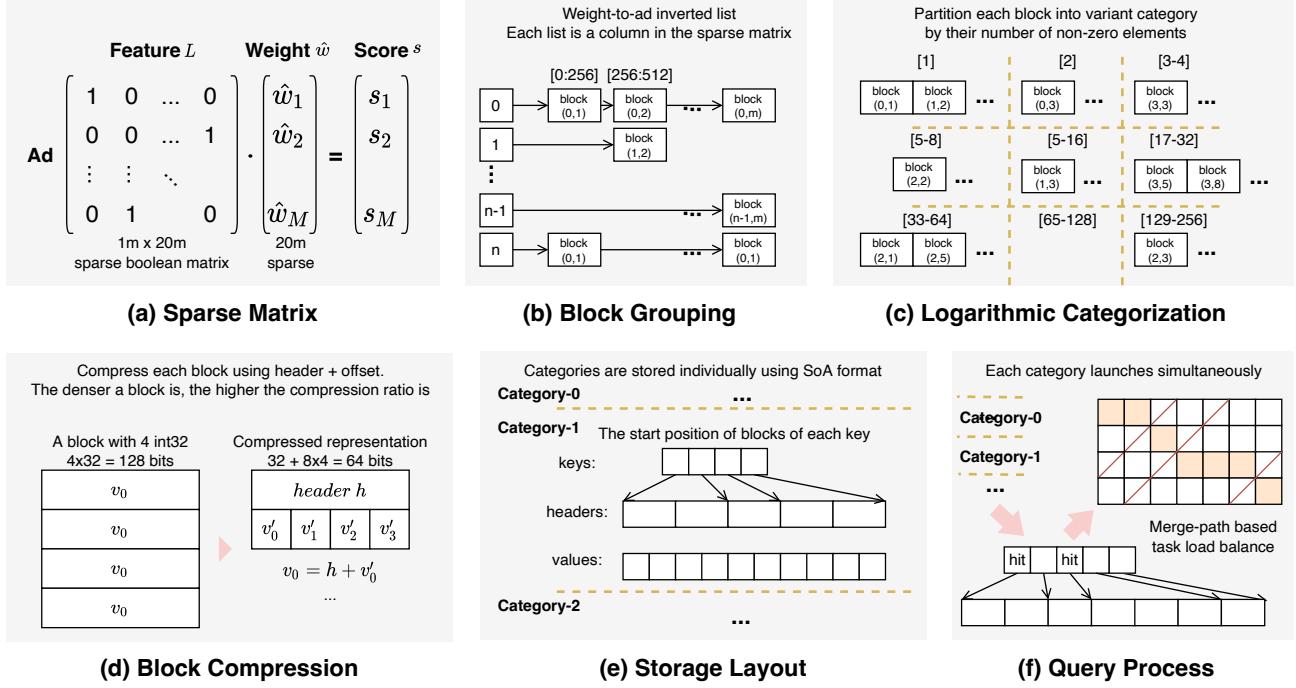


Figure 3: The overall proceed of the proposed GPU-powered HitMatch operator

of sparse matrix operations impractical in real-world scenarios requiring high throughput and low latency.

To overcome this limitation, we observe that the sparse matrix L consists solely of binary values and remains constant across different queries. These properties allow us to preprocess L into an optimized index prior to any query execution. Specifically, we recognize the binary matrix L is naturally in the form of an inverted list. Furthermore inverted lists are widely used in various fields and are well-suited for GPU processing [41, 26, 5]. Consequently, we propose a novel GPU-powered inverted list data structure to preprocess the matrix L .

Our design is driven by two key considerations: (a) **Load balance**: the computation is done on GPU, so the load balance is essential for the performance, and (b) **Reducing IO**: the sparse matrix multiplication or inverted list reduction is considered as an IO-bound operator, so reducing the size of data to be processed is another aspect to achieve high performance. With these two considerations, we represent the matrix L as an compressed inverted list, where each list represents a column in L , the keys are the feature indices, and the values are the ad indices. The processing proceeds as follows:

Block Grouping: To ensure efficient storage and computation, we first split lists into blocks by the lowest 8 bits of ad indices and consider each block as a basic unit of the index in the following process. This ensures uniform block sizes, which make it possible to further compress the data storage.

Logarithmic Categorization: Next, we partition blocks into variant groups according to their number of ads in each block. For a block containing n ads, it will be partitioned into the $\lceil \log_2(n) \rceil$ -th

group. Then, we pad each block into the same length of $2^{\lceil \log_2(n) \rceil}$. Hence, the block in the same group will have the similar workload, which is essential for the load-balance.

Block Compression: Once the blocks are grouped, we compress each block to minimize storage requirements. Specifically, for each block, we utilize only one header value representing the same highest 24 bits for all the ads in the block so we can store an ad in a block by only extra 8 bits integers by their lower 8-bits. This minimizes the size of data storage, significantly alleviating IO bottlenecks.

Storage Layout: We store the blocks in a struct-of-arrays (SoA) format, enhancing memory coalescing during GPU computation [1]. In order to locate the begin and end of the data given each key efficiently, for keys array, we store the segment of each key indicating the offset of the each key.

The overall processing pipeline of overall indexing procedure is visualized in Figure 3, and the detailed implementation is outlined in Algorithm 1.

When a query comes, each block group is computed in parallel separately. For each non-zero feature value x of query, we lookup all the blocks using its index and add the score for each ad by \hat{w}_i . To further improve the load balance, we integrate the merge-based load balance algorithm [3]. This approach dynamically assigns computational tasks to GPU threads according to the number of blocks within each list for the current block group. Specifically, our approach first locates the starting and ending positions of target data entries within the stored key array based on the provided keys. We then compute the prefix sum of data lengths per key using a parallel exclusive scan algorithm[30] to determine data

offsets. Subsequently, a merge-based load balancing strategy dynamically assigns tasks to all GPU threads in parallel, utilizing the precomputed keys and offsets to ensure optimal thread utilization. By adjusting the workload distribution in real-time, our method further handles imbalances caused by uneven block sizes. Figure 3 (d) shows an overview of the HitMatch operator implementation, and the query processing algorithm is formalized in Algorithm 2.

In our advertising retrieval system, the feature-to-ad mapping matrix updates every few minutes. Upon receiving a matrix update, the server immediately constructs a GPU-accelerated inverted list via Algorithm 1, and use this structure to accelerate the subsequent explicit feature interaction computations. Given that a single built index can serve hundreds of thousands of queries, the associated indexing overhead is effectively amortized due to its reusability across numerous inference requests.

Algorithm 1 Indexing Algorithm

Input: Input key-value map $L \triangleright$ keys: feature indices, values: ad indices

- 1: $B \leftarrow$ Empty hash map with default value \emptyset ▷ Blocks
- 2: $G \leftarrow$ Empty hash map with default value \emptyset ▷ Groups
- 3: **for all** $(k, v) \in L$ **in parallel do** ▷ Step 1: Block Grouping and Compression
- 4: $h, l \leftarrow \lfloor v/2^8 \rfloor, v \bmod 2^8$
- 5: $B[k, h] \leftarrow B[k, h] \cup \{l\}$
- 6: **end for**
- 7: **for all** $(k, h), ls \in B$ **in parallel do** ▷ Step 2: Group Partitioning
- 8: $n \leftarrow |ls|$
- 9: $g \leftarrow \lceil \log_2(n) \rceil$
- 10: Pad ls with 0 to length 2^g
- 11: $G[g, k, h] \leftarrow G[g, k, h] \cup \{ls\}$
- 12: **end for**
- 13: Allocate arrays keys, header, values for each group in G ▷ Step 3: Struct-of-Array (SoA) Storage
- 14: **for** $g \in \{0, 1, \dots, 8\}$ **in parallel do**
- 15: $i \leftarrow 0$
- 16: Append 0 to $\text{keys}[g]$
- 17: **for all** $(k, h), ls \in G[g]$ **do**
- 18: Append h to $\text{header}[g]$
- 19: Append ls to $\text{values}[g]$
- 20: **while** $i < k$ **do**
- 21: Append k to $\text{keys}[g]$
- 22: $i \leftarrow i + 1$
- 23: **end while**
- 24: **end for**
- 25: **end for**
- 26: **return** keys, header, values

4 Experiments

In this section, we conduct experiments to evaluate our proposed methods, aiming to answer the following key research questions:

RQ1: Does our method achieve superior offline state-of-the-art (SOTA) performance compared to standard baseline models?

Algorithm 2 Query Algorithm

Input: Query q with non-zero feature values $\{(k, \hat{w})\}$, Indexed data in SoA format: {keys, header, values}

- 1: Initialize scores as a zero array of size equal to the number of ads
- 2: **for** $g \in \{0, 1, \dots, 8\}$ **in parallel do**
- 3: $k_length \leftarrow [\text{keys}[g, k_i + 1] - \text{keys}[g, k_i]] \text{ for } k_i \in q]$
- 4: $k_seg \leftarrow \text{ExclusiveScan}(k_length)$
- 5: **for all** $k, k_offset \in \text{LoadBalance}(k_seg)$ **in parallel based on [3] do**
- 6: $h \leftarrow \text{header}[g, k_offset]$
- 7: $ls \leftarrow \text{values}[g, k_offset]$
- 8: **for all** $l \in ls$ **do**
- 9: $\text{AtomicAdd}(\text{scores}[h \cdot 2^8 + l], \hat{w}[k])$
- 10: **end for**
- 11: **end for**
- 12: **end for**
- 13: **return** scores

RQ2: Can our method significantly enhance online advertising revenue?

RQ3: Does our method demonstrate more efficient resource utilization in terms of engineering performance compared to existing sparse matrix multiplication methods?

4.1 Experimental Setup

4.1.1 Datasets. We collect historical advertising data to evaluate our proposed models from the WeChat-channel video feed, a prominent short video sharing platform within WeChat. The reason why we do not choose publicly available benchmark in this study is due to the critical need for evaluating the proposed GPU Hitmatch operator on a large-scale advertisement candidate pool. To meet these demands, we gathered seven days of ad request logs to evaluate our proposed algorithms. The first six days' logs were used for training dataset, while the seventh day's data were reserved for testing dataset. To measure the engineering performance metrics of the model's inference capabilities, we utilized the online advertisement candidate pool containing millions of instances, enabling a thorough evaluation of the system's scalability and operational efficiency.

4.1.2 Features. Our model leverages user-side features categorized into basic user profiles and historical behavioral data organized in sequence. The former encompasses demographic and the latter records user interactions with the ad videos, including views, follows, likes, clicks, and other engagement metrics. Additionally, the ad features include material attributes (e.g., title, description, images, and videos), hierarchical categories, bidding price, optimization goals and other meta data.

In this study, the cross-features including the hierarchical categories, advertiser and other meta data from the advertisements that users have previously interacted with as the input for the HitMatch operator. Such features are statistically aggregated dynamically in real time, including the click counts, click-through rates, conversion counts, and conversion rates. The aggregation time windows are performed in multiple period, such as 1, 7, 15, 30 days and so on,

Table 1: Overall performance comparison on the different methods in offline evaluation

Method	GAUC	Recall@5_1	Recall@10_1
DT	0.839	0.730	0.908
DT-IPNN	0.843	0.739	0.913
DT-HitMatch	0.847	0.753	0.924
DT-IPNN-HitMatch	0.861	0.768	0.939
Improvement over DT	2.62%	5.21%	3.41%

encompassing advertisements that users have engaged with during this period.

4.1.3 Compared Models and Evaluation Metrics. In our evaluation, we perform a thorough analysis of various models, including the baseline dual-tower model (DT) and the proposed DT-IPNN-HitMatch. To ensure a comprehensive comparison, we also conduct additional experiments with the dual-tower model integrated with the IPNN module (DT-IPNN) and the HitMatch module (DT-HitMatch). We mainly evaluate the following metrics:

GAUC: This metric computes the weighted average of AUC values at the user level, where the top-1 ad is designated as the positive instance and the remaining as negative.

$$GAUC = \frac{1}{\#pv} \sum_{i=1}^{\#pv} AUC_i \quad (10)$$

where #pv denotes number of requests.

Recall@G_k: This metric quantifies the proportion of top-k ranked ads in the final ranking that are present within the top-G retrieval ads, with higher values indicating superior performance. As only one ad is displayed per request in our scenario and we only sample 30 ranked ads per request for training model. Thus, we primarily focuses on the case where k = 1, G ∈ [5, 10] in offline evaluations and G ∈ [100] in online evaluations.

$$\text{Recall}@G_k = \frac{1}{\#pv} \sum_{i=1}^{\#pv} \frac{|\{\text{Top}G \text{ ads}\} \cap \{\text{Top}K \text{ ads}\}|}{k} \quad (11)$$

Online metrics: For online metrics, we mainly care about advertising cost and GMV (Gross Merchandise Volume)

Efficiency metrics: Maximum QPS and pre-processing time for our inverted list index for the HitMatch operator.

We train the aforementioned models using the LTR paradigm, with sampling 30 ads in the final ranking phase per request. The model is trained online in real-time with a batch size of 4000. User and ad feature embeddings are 32-dimensional. Dense DNN parameters are optimized using Adam (learning rate: 0.015, beta: 1.0), while sparse parameters use FTRL (learning rate: 1.0E-4, beta_1: 0.5, beta_2: 0.999, epsilon: 1.0E-8). Both user and ad towers produce 64-dimensional vectors using DNN.

4.2 Offline Evaluations (RQ1)

The offline experimental evaluation results, as summarized in Table 1, provide a detailed comparison of the performance of our proposed models against the baseline DT model across three key metrics: GAUC, Recall@5_1, and Recall@10_1. The results clearly indicate

Table 2: The benchmark of the explicit feature interaction operation

Method	Preprocessing time (ms)	QPS (1/s)
cuSPARSE	N.A.	275.94
Ours	224.32	1904.23
Improvement	N.A.	590%

that all three proposed models—DT-IPNN, DT-HitMatch, and DT-IPNN-HitMatch—consistently outperform the baseline DT model, with DT-IPNN-HitMatch achieving the highest overall performance. Specifically, DT-IPNN-HitMatch demonstrates significant improvements, achieving a GAUC of 0.861, a Recall@5_1 of 0.768, and a Recall@10_1 of 0.939, which represent relative gains of 2.62%, 5.21%, and 3.41%, respectively, over the baseline DT model. The superior performance of DT-HitMatch and DT-IPNN-HitMatch compared to DT-IPNN and DT can be attributed to the incorporation of explicit cross-features, which serve as strong signals in our system. These cross-features effectively capture the interactions between user preferences and item characteristics, leading to more accurate recommendations. Additionally, the stability of user interests over a given time period further enhances the effectiveness of these features, as they allow the model to better align recommendations with long-term user preferences.

4.3 HitMatch Operator Evaluations (RQ2)

To illustrate the performance of the proposed inverted list index for the explicit feature interaction, we also conduct the experiment against the cuSPARSE [31] library, a GPU implementation of the sparse matrix multiplication provided by Nvidia. We collect the sparse matrix L from the historical advertising data and utilize 1000 feature vectors as queries to benchmark the time usage between cuSPARSE and the proposed approach on an instance with a Nvidia Tesla T4 graphic card. The results are shown in Table 2. As can be seen, our method outperforms the cuSPARSE by around 7 times in QPS(+590%). This shows that the proposed approach indeed have superior performance owing to the lower data transformation and better load balance by introducing a small preprocessing time. The preprocessing time of our method is 224 ms, which is negligible amortized by a large number of queries.

4.4 Online Evaluations (RQ3)

We deployed our proposed models using the aforementioned feature settings in the Tencent Advertising online system with A/B testing. The proposed model demonstrated significant performance improvements as shown in table 3, achieving cost, GMV and Recall@100_1 increase by 0.37%, 1.58% and 1.8% respectively in the WeChat Moment. With 1.25%, 1.49% and 2.5% increase in cost, GMV and Recall@100_1 respectively in WeChat Channel. It is worth to noting that the performance is closely tied to the design of cross-feature types and their quantity in practical application. Generally, richer and more diverse cross-features may lead to improved results. For millions of ads in WeChat Moments and Channel, the proposed HitMatch operator can be done within 500 us on a Nvidia

Table 3: Improvement in online evaluation by A/B Test

Traffic	cost	GMV	Recall@100_1
WeChat-Moment	+0.37%	+1.58%	+1.8%
WeChat-Channel	+1.25%	+1.49%	+2.5%

T4 GPU, demonstrating its ability to efficiently address explicit feature interaction problem at an industry scale.

5 Conclusion

In this paper, we propose a novel approach from a model-engineering co-design perspective to integrating both implicit and explicit feature interactions for ad retrieval models, leveraging a GPU-accelerated inverted list index. Our method significantly reduces computational overhead, enabling real-time processing of millions of ads in our platforms such as WeChat Moments and Channel. Our extensive experiments on real-world datasets demonstrate substantial improvements in recommendation accuracy and query latency, outperforming the baseline by a large margin. These results demonstrate the effectiveness of our method, and providing a practical insight for optimizing large-scale ad retrieval systems.

References

- [1] Sara S Baghsorkhi, Isaac Gelado, Matthieu Delahaye, and Wen-mei W Hwu. 2012. Efficient performance evaluation of memory hierarchy for highly multi-threaded graphics processors. *ACM SIGPLAN Notices*, 47, 8, 23–34.
- [2] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. 2003. Web search for a planet: the google cluster architecture. *IEEE micro*, 23, 2, 22–28.
- [3] Sean Baxter. Moderngpu 2.0. [https://github.com/moderngpu/moderngpu/wiki,\(2016\).](https://github.com/moderngpu/moderngpu/wiki,(2016).)
- [4] Weiwei Bian et al. 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*, 57–65.
- [5] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient inverted indexes for approximate retrieval over learned sparse representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 152–162.
- [6] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: an overview. *Learning*, 11, 23–581, 81.
- [7] Rihan Chen et al. 2022. Approximate nearest neighbor search under neural similarity metric for large-scale recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 3013–3022.
- [8] Heng-Tze Cheng et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, 7–10.
- [9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, 191–198.
- [10] Doug Cutting and Jan Pedersen. 1989. Optimization for dynamic inverted index maintenance. In *Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, 405–411.
- [11] Jeffrey Dean et al. 2009. Challenges in building large-scale information retrieval systems. In *Keynote of the 2nd ACM international conference on web search and data mining (WSDM)* number 1498759.1498761. Vol. 10.
- [12] Ali Mamdouh Elkahlky, Yang Song, and Xiaodong He. 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th international conference on world wide web*, 278–288.
- [13] Weihao Gao et al. 2020. Deep retrieval: learning a retrievable structure for large-scale recommendations. *arXiv preprint arXiv:2007.07203*.
- [14] Huifeng Guo, Ruiming Tang, Yuning Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*.
- [15] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. <https://arxiv.org/abs/1908.10396>.
- [16] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate nearest neighbor: towards removing the curse of dimensionality.
- [17] 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.
- [18] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2333–2338.
- [19] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33, 1, 117–128.
- [20] Jeff Johnson, Matthijs Douze, and Hervé Jegou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7, 3, 535–547.
- [21] 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.
- [22] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.
- [23] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony KH Tung. 2020. Locality-sensitive hashing scheme based on longest circular co-substring. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2589–2599.
- [24] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience*, 45, 1, 1–29.
- [25] Honghao Li, Yiwen Zhang, Yi Zhang, Hanwei Li, Lei Sang, and Jieming Zhu. 2024. Dcnv3: towards next generation deep cross network for ctr prediction. *arXiv preprint arXiv:2407.13349*.
- [26] Minghan Li, Sheng-Chieh Lin, Xueguang Ma, and Jimmy Lin. 2023. Slim: sparsified late interaction for multi-vector retrieval with inverted indexes. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1954–1959.
- [27] Juxin Lin et al. 2024. Enhancing relevance of embedding-based retrieval at walmart. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 4694–4701.
- [28] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3, 3, 225–331.
- [29] 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, 824–836.
- [30] Duane Merrill and Michael Garland. 2016. Single-pass parallel prefix scan with decoupled look-back. *NVIDIA, Tech. Rep. NVR-2016-002*.
- [31] Maxim Naumov, L Chien, Philippe Vandermersch, and Ujval Kapasi. 2010. Cusparse library: a set of basic linear algebra subroutines for sparse matrices. In *GPU Technology Conference* number 5. Vol. 3, 17.
- [32] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 world wide web conference*, 1349–1357.
- [33] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 1149–1154.
- [34] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining*. IEEE, 995–1000.
- [35] Anil Shanbhag, Bobbi W Yogatama, Xiangyao Yu, and Samuel Madden. 2022. Tile-based lightweight integer compression in gpu. In *Proceedings of the 2022 International Conference on Management of Data*, 1390–1403.
- [36] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, 101–110.
- [37] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. Bert4rec: sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, 1441–1450.
- [38] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, 1–7.
- [39] Zhe Wang, Lijun Zhao, Biye Jiang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. Cold: towards the next generation of pre-ranking system. *arXiv preprint arXiv:2007.16122*.
- [40] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: a survey and new perspectives. *ACM computing surveys (CSUR)*, 52, 1, 1–38.
- [41] Jingbo Zhou, Qi Guo, HV Jagadish, Lubos Krcal, Siyuan Liu, Wenhao Luan, Anthony KH Tung, Yueji Yang, and Yuxin Zheng. 2018. A generic inverted index framework for similarity search on the gpu. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 893–904.

- [42] 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.
- [43] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-scalar ram-cpu cache compression. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 59–59.