MTGR: Industrial-Scale Generative Recommendation Framework in Meituan

Ruidong Han
Meituan
Beijing, China
hanruidong@meituan.com

Bin Yin
Meituan
Beijing, China
yinbin05@meituan.com

Shangyu Chen
Meituan
Beijing, China
chenshangyu03@meituan.com

He Jiang
Meituan
Beijing, China
jianghe06@meituan.com

Fei Jiang
Meituan
Beijing, China
jiangfei05@meituan.com

Xiang Li
Meituan
Beijing, China
lixiang245@meituan.com

Chi Ma
Meituan
Beijing, China
machi04@meituan.com

Mincong Huang
Meituan
Beijing, China
huangmincong@meituan.com

Xiaoguang Li
Meituan
Beijing, China
lixiaoguang12@meituan.com

Chunzhen Jing
Meituan
Beijing, China
jingchunzhen@meituan.com

Yueming Han
Meituan
Beijing, China
hanyueming02@meituan.com

Menglei Zhou
Meituan
Beijing, China
zhoumenglei@meituan.com

Lei Yu
Meituan
Beijing, China
yulei37@meituan.com

Chuan Liu
Meituan
Beijing, China
liuchuan11@meituan.com

Wei Lin
Meituan
Beijing, China
linwei31@meituan.com

Abstract

Scaling law has been extensively validated in many domains such as natural language processing and computer vision. In the recommendation system, recent work has adopted generative recommendations to achieve scalability, but their generative approaches require abandoning the carefully constructed cross features of traditional recommendation models. We found that this approach significantly degrades model performance, and scaling up cannot compensate for it at all. In this paper, we propose MTGR (Meituan Generative Recommendation) to address this issue. MTGR is modeling based on the HSTU [22] architecture and can retain the original deep learning recommendation model (DLRM) features, including cross features. Additionally, MTGR achieves training and inference acceleration through user-level compression to ensure efficient scaling. We also propose Group-Layer Normalization (GLN) to enhance the performance of encoding within different semantic spaces and the dynamic masking strategy to avoid information leakage. We further optimize the training frameworks, enabling support for our models with 10 to 100 times computational complexity compared to the DLRM, without significant cost increases. MTGR achieved 65x FLOPs for single-sample forward inference compared to the DLRM model, resulting in the largest gain in nearly two years both offline and online. This breakthrough was successfully deployed on Meituan, the world's largest food delivery platform, where it has been handling the main traffic.

CCS Concepts

• **Information systems** → **Recommendation systems**.

Keywords

Scaling Law; Generative Recommendation

ACM Reference Format:
Ruidong Han, Bin Yin, Shangyu Chen, He Jiang, Fei Jiang, Xiang Li, Chi Ma, Mincong Huang, Xiaoguang Li, Chunzhen Jing, Yueming Han, Menglei Zhou, Lei Yu, Chuan Liu, and Wei Lin. 2025. MTGR: Industrial-Scale Generative Recommendation Framework in Meituan. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Scaling law have been proven to apply to most deep learning tasks, including language models[9], computer vision[14, 23], and information retrieval[4]. Under the requirements of high QPS (Queries Per Second) and low latency in industrial recommendation systems,

the scaling of the model is usually limited by both the training cost and inference time. Currently, the study on scaling ranking models can be divided into two types: Deep Learning Recommendation Model (DLRM) and Generative Recommendation Model (GRM). DLRM models individual user-item pairs to learn the probability of interest for ranking and scales up by developing more complex mappings. GRM organizes data by token like natural language and performs next token prediction through a transformer architecture.

In industrial recommendation systems, DLRM has been used for nearly a decade, typically with inputs that include a large number of well-designed handcrafted features such as cross features [1] to improve model performance. However, DLRM has two significant drawbacks when it comes to scaling: 1) with exponential growth of user behavior, traditional DLRM cannot efficiently process entire user behaviors, often resorting to sequence retrieval or designing low-complexity modules for learning, which limits the model's learning capability [1, 15]. 2) scaling based on DLRM incurs approximately linear costs in training and inference with the number of candidates, making the expenses unbearably high.

For GRM, recent studies have pointed out its excellent scalability [3, 22]. We identify two key factors: 1) GRM directly models the complete chain of user behavior, which compresses multiple samples of exposure under the same user into one. This significantly reduces computational redundancy, while allowing end-to-end encoding of longer sequence compared to DLRM; 2) GRM adopts a transformer architecture with efficient attention computation[2, 22], enabling the model's training and inference to meet the requirements of industrial recommendation systems. However, GRM heavily relies on next token prediction to model a complete user behavior sequence, which requires removing cross features between candidates and the user. We found that excluding cross features severely damages the model's performance, and this degeneration cannot be compensated by scaling up at all.

How can we build a ranking model that utilizes cross features to ensure effectiveness while also possessing the scalability of GRM? To address this problem, we propose the Meituan Generative Recommendation (MTGR). Compared to traditional DLRM and GRM, MTGR utilizes the advantages and discards disadvantage of the methods. MTGR maintains inputs feature consistent with traditional DLRM, including cross features. Specially, MTGR reorganizes features by converting user and candidate features into different tokens, leading to a token sequence for efficient model scaling. Then MTGR incorporates the cross feature in the candidate tokens and learns with a discriminative loss.

MTGR employs the similar HSTU (Hierarchical Sequential Transduction Units) architecture as used in [22] for modeling. In HSTU, we propose Group-layer Normalization(GLN) to normalize different types of token separately, enabling better modeling of multiple heterogeneous information simultaneously. In addition, we propose a dynamic masking strategy, where full-attention, auto-regressive and visibility only to itself are used to ensure performance and avoid information leakage.

Unlike the commonly used TensorFlow in the industry, MTGR training framework is built based on TorchRec[8] and optimized for computational efficiency. Specifically, to handle the real-time insert/delete of sparse embedding entries, we employ dynamic hash tables to replace static tables. To improve efficiency, we conduct dynamic sequence balancing to address the computation load imbalances among GPUs and adopt embedding ID de-duplication alongside automatic table merging to accelerate embedding lookup. We also incorporate implementation optimization including mixed precision training and operator fusion. Compared to TorchRec, our optimized framework improves training throughput by 1.6x – 2.4x while achieving good scalability when running over 100 GPUs.

We validate the scalability of MTGR on a small-scale dataset. Then, we design three models of varying sizes, which are trained using following over six months of data, to verify the scaling law for both offline and online performance. The large version, compared to the DLRM baseline optimized over years, demonstrates 65x FLOPs per sample for forward computation and increases the conversion volumes by 1.22% and the CTR (Click-Through Rate) by 1.31% in our scenarios. Meanwhile, the training cost remains unchanged and the inference cost is reduced by 12%. MTGR-large has been deployed in the Meituan take-away recommendation system, serving hundreds of millions of users.

In summary, our contributions are summarized as follows:

- MTGR combines the advantages of DLRM and GRM, retaining all the features of DLRM, including the cross feature, while demonstrating excellent scalability as GRM.
- We propose Group-Layer Normalization and dynamic masking strategies to achieve better performance.
- We perform systematic optimizations on the TorchRec-based MTGR training framework to enhance training performance.
- Both offline and online experiments were conducted to demonstrate the power-law relationship between the performance of MTGR and computational complexity, and its superiority compared to DLRM.

2 Related Works

2.1 Deep Learning Recommendation Model

A classic DLRM structure typically includes many inputs such as context (e.g. time, location), user profile (e.g. gender, age), user behavior sequences, and target item with many cross features. Two particularly important modules in ranking model are behavior sequence processing and feature interactions learning. The behavior sequence module[15, 17, 26] usually employs target attention mechanisms to capture the similarity between the historical behavior of the users and the item to be estimated. The feature interactions module[12, 18–20] is designed to capture interactions among different features including user and item to produce the final prediction.

2.2 Scaling up Recommendation Model

Based on the different scaling modules in DLRM, there are two distinct approaches. One approach is the scaling cross module, i.e. scaling up the feature interactions module that integrates user and item information. [24] introduces a stackable Wukong layer for scaling up. [6] employ a multi-embedding strategy to address the embedding collapse, thereby enhancing the model's scalability. The other approach is the scaling user module, where only the user part is scaled up, making this approach more inference-friendly.

[1]Cross features measure the interactions of multiple raw features, like the user's historical click-through rate for the target candidate

[7, 25] reduce online inference costs by scaling only user representations and caching or broadcasting them to different items that will be estimated. [16, 21] design a pre-training method for user representations, demonstrating scalability in downstream tasks.

The counterpart to DLRM is GRM. [22] validate the scaling law through HSTU with scaling up to trillion-level parameters. [3] use semantic coding to replace traditional ID representations, combining DPO optimization with a transformer-based framework to replace the cascaded learning framework with a unified generative model.

3 Preliminary

3.1 Data Arrangement

Traditionally, for a user and the corresponding K candidates, the i-th sample for the user and i-th candidate pair can be represented as $\mathbb{D}_i = [\mathbf{U}, \overrightarrow{\mathbf{S}}, \overrightarrow{\mathbf{R}}, \mathbf{C}_i, \mathbf{I}_i]$. Specifically, $\mathbf{U} = [\mathbf{U}^1, ..., \mathbf{U}^{N_\mathbf{U}}]$ represents the user's profile feature ($\mathbf{U}^i$), such as age, gender, etc. Each feature $\mathbf{U}^i$ is a scalar, and $N_\mathbf{U}$ represents the number of features used. $\overrightarrow{\mathbf{S}} = [\mathbf{S}^1, ..., \mathbf{S}^{N_\mathbf{S}}]$ contains the sequence of items that have historically been interacted with by the user. Each element in $\mathbf{S}^i = [\mathbf{s}^1, ..., \mathbf{s}^{N_s}]$ represents an item, it is comprised by selected features ($\mathbf{s}^i$) such as the item's ID, tag, average CTR on the item and etc. Similar to $\overrightarrow{\mathbf{S}}$, $\overrightarrow{\mathbf{R}}$ records the closet interaction to current request within a few hours or a day. $\overrightarrow{\mathbf{R}}$ represents user's real-time actions and preference. It shares the same features with $\overrightarrow{\mathbf{S}}$. $\mathbf{C} = [\mathbf{C}^1, ..., \mathbf{C}^{N_\mathbf{C}}]$ comprises the cross features between the user and the candidates. $\mathbf{I} = [\mathbf{I}^1, ..., \mathbf{I}^{N_\mathbf{I}}]$ contains the features used for candidates, such as item's ID, tag, and brand. $\mathbf{I}$ is candidate-dependent and shared for different users.

3.2 Ranking Model in Recommendation Systems

With input samples $\mathbb{D}$, traditional recommendation system processes the samples independently. Specially, it firstly embeds the features in $\mathbb{D}$ and converts the samples to a dense representation. Formally, the features in $\mathbf{U}$, $\mathbf{C}$, $\mathbf{I}$ are embedded and concatenated to $\mathbf{Emb}_\mathbf{U} \in \mathbb{R}^{K \times d_\mathbf{U}}$, $\mathbf{Emb}_\mathbf{C} \in \mathbb{R}^{K \times d_\mathbf{C}}$ and $\mathbf{Emb}_\mathbf{I} \in \mathbb{R}^{K \times d_\mathbf{I}}$, respectively. For features in $\overrightarrow{\mathbf{S}}$ and $\overrightarrow{\mathbf{R}}$ [2], each item($\mathbf{S}^i$)'s feature are similarly embedded and concatenated to $\mathbf{Emb}_{\mathbf{S}^i} \in \mathbb{R}^{d_\mathbf{s}}$, items in $\overrightarrow{\mathbf{S}}$ are concatenated in other dimension, leading to $\mathbf{Emb}_{\overrightarrow{\mathbf{S}}} \in \mathbb{R}^{N_{\overrightarrow{\mathbf{S}}} \times d_\mathbf{s}}$ [3].

To extract user interest between the historical interacted items and candidates, target attention normally be used with target as query and sequence feature as key/value. Formally,

$$\mathbf{Feat}_{\overrightarrow{\mathbf{S}}} = \text{Attention}(\mathbf{Emb}_\mathbf{I}, \mathbf{Emb}_{\overrightarrow{\mathbf{S}}}, \mathbf{Emb}_{\overrightarrow{\mathbf{S}}}) \in \mathbb{R}^{K \times d_\mathbf{S}} \tag{1}$$

Eq.1 aggregates $\overrightarrow{\mathbf{S}}$ according to $\mathbf{I}$. Finally, embedded and processed features from $\mathbb{D}$ are concatenated and represented as:

$$\mathbf{Feat}_\mathbb{D} = [\mathbf{Emb}_\mathbf{U}, \mathbf{Feat}_{\overrightarrow{\mathbf{S}}}, \mathbf{Feat}_{\overrightarrow{\mathbf{R}}}, \mathbf{Emb}_\mathbf{C}, \mathbf{Emb}_\mathbf{s}] \in \mathbb{R}^{K \times (d_\mathbf{U} + d_\mathbf{S} + d_\mathbf{C} + d_\mathbf{I})} \tag{2}$$

$\mathbf{Feat}_\mathbb{D}$ is further fed to a multiple layer perceptron (MLP) to output logit for each sample. The logit is used for learning in training and for ranking when inference.

[2] In the following description, since $\overrightarrow{\mathbf{S}}$ and $\overrightarrow{\mathbf{R}}$ are processed in a similar way, only $\overrightarrow{\mathbf{S}}$'s modeling is described for clarity.

[3] N_* represents the sequence length of $*$

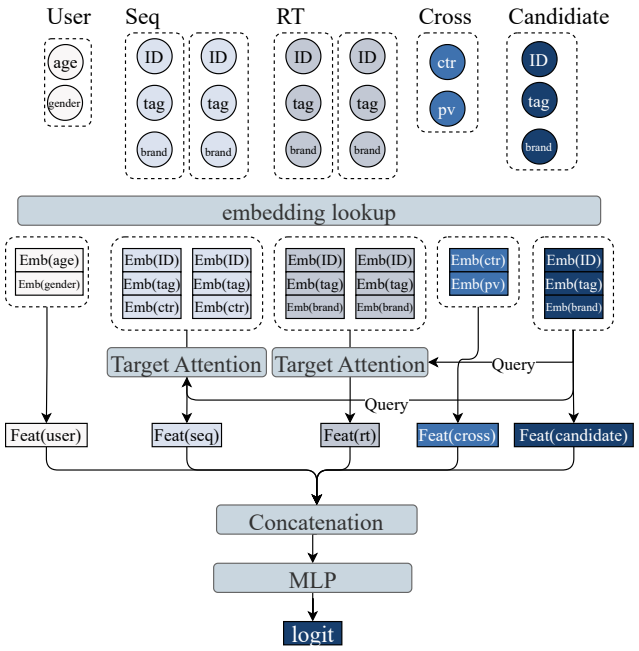

Figure 1: Data Arrangement and workflow of traditional ranking model. An example with simplified features is demonstrated: U contains 'age' and 'gender'; $\overrightarrow{\mathbf{S}}$ and $\overrightarrow{\mathbf{R}}$ is comprised by 2 item respectively, each item possesses 'ID', 'tag' and 'brand'; C contains 'ctr' and 'pv', presenting user's historical CTR and number of exposures to the candidate, which uses 'ID', 'tag' and 'brand'.

Fig.1 shows a simplified data arrangement and workflow under the traditional ranking model. These features are firstly embedded, the leading embeddings are processed in different methods. Finally, the processed features are concatenated and processed by MLP for feature interaction. The final logit is generate for each candidates.

3.3 Scaling Dilemma in Recommendation Systems

Model scaling has been a common method for performance improvement in ranking. Generally, model scaling aims at scaling parameters in user module and cross module. The user module processes user feature including sequence features and generates user-dependent representation. Scaling user module brings in better representation for user. Besides, since the user is shared and inferred once for all candidates, a large inference cost in the user module does not lead to a burdensome system overload. However, scaling user module alone does not contribute to feature interaction in user and item directly.

On the contrary, another trend of method aims at scaling cross module, i.e., the feature interaction MLP after feature concatenation. These methods enhance ranking ability by paying more attention to interaction between user and candidates. However, since cross module is inferred for each candidates, computation increasing is scaled linearly with number of candidates. Cross module scaling brings in unacceptable system latency.

The scaling dilemma in traditional recommendation system necessitates a new scaling method, leading to efficient feature interaction between user and candidates, while at the same time brings sub-linear inference cost with number of candidates. MTGR innovates the scaling approach in recommendation system by data rearrangement and corresponding architecture optimization.

4 Data Rearrangement and Architecture of MTGR

4.1 User Sample Aggregation for Training and Inference Efficiency

Compared with feature categorization in Sec.3.1, for i-th sample in candidates, MTGR organizes features into $\mathbb{D}_i = [\mathbf{U}, \overrightarrow{\mathbf{S}}, \overrightarrow{\mathbf{R}}, [\mathbf{C}_i, \mathbf{I}_i]]$. Specially, the cross feature $\mathbf{C}$ is arranged as part of the item feature of candidates. In MTGR, candidates are aggregated by user in a specific window for training and by request for inference. Since the aggregation is performed by the same user, the aggregated sample can use the same user representation ($\mathbf{U}, \overrightarrow{\mathbf{S}}, \overrightarrow{\mathbf{R}}$). In particular, $\overrightarrow{\mathbf{R}}$ arranges all the user's real-time interaction items within another specific window in chronological order of interaction time.

Fig.2(a) demonstrates the aggregation: Compared with Fig.1 where only one candidate is predicted, Fig.2(a) aggregates three items in one sample, reusing the same user representation. Formally, it forms a feature representation for the same user:

$$\mathbb{D} = [\mathbf{U}, \overrightarrow{\mathbf{S}}, \overrightarrow{\mathbf{R}}, [\mathbf{C}, \mathbf{I}]_1, ..., [\mathbf{C}, \mathbf{I}]_K] \tag{3}$$

By aggregating candidates into one sample, MTGR reduces a tremendous amount of resource by performing only a single computation and producing scores for all candidates. Specially, the user aggregation procedure greatly reduce the training samples from the sizes of all candidates to all users. For inference, the candidates in a request are grouped as mentioned above, MTGR only perform one-time inference for all the candidate ranking, instead of inference of candidates' number. The aggregation circumvents the dependency of candidate' volume in inference cost, leaving possibility and potential for model scaling.

Eq.3 is a combination of scalar and sequence features. To unify the input format, MTGR converts features and sequence into tokens. Specially, for scalar features in $\mathbf{U}$, each feature is naturally converted to individual token with dimension $\mathbf{Feat}_\mathbf{U} \in \mathbb{R}^{N_\mathbf{U} \times d_{\text{model}}}$. d_{model} is a set unified dimension for all tokens. For sequence feature of $\overrightarrow{\mathbf{S}}$ and $\overrightarrow{\mathbf{R}}$, each item $\mathbf{S}$ is regarded as a token. Features in $\mathbf{S}$ is firstly embedded and concatenated, then a MLP module is adopted for dimension unification. Formally, $\mathbf{Feat}_{\mathbf{S}_i} = \text{MLP}(\text{Concat}(\mathbf{Emb}_\mathbf{s})) \in \mathbb{R}^{d_{\text{model}}}$. The features of $\mathbf{S}$ in the sequence is concatenated along another dimension, leading to $\mathbf{Feat}_{\overrightarrow{\mathbf{S}}} \in \mathbb{R}^{N_{\overrightarrow{\mathbf{S}}} \times d_{\text{model}}}$.

Similar, each item $\mathbf{I}$ in candidates is regarded as a token. The features in candidates are embedded and concatenated, converted to the unified dimension by another MLP. Candidates are concatenated as a series of tokens: $\mathbf{Feat}_\mathbf{I} = \text{Concat}(\text{MLP}(\text{Concat}(\mathbf{Emb}_{\mathbf{C}_i}, \mathbf{Emb}_{\mathbf{I}_i}))) \in \mathbb{R}^{N_\mathbf{I}}$ Finally, the constructed tokens from $\mathbf{U}$, $\overrightarrow{\mathbf{S}}$, $[\mathbf{C}, \mathbf{s}]$ are concatenated, leading a long sequence of tokens:

$$\begin{aligned} \mathbf{Feat}_\mathbb{D} &= \text{Concat}([\mathbf{Feat}_\mathbf{U}, \mathbf{Feat}_{\overrightarrow{\mathbf{S}}}, \mathbf{Feat}_{\overrightarrow{\mathbf{R}}}, \mathbf{Feat}_\mathbf{I}]) \\ &\in \mathbb{R}^{(N_\mathbf{U} + N_{\overrightarrow{\mathbf{S}}} + N_{\overrightarrow{\mathbf{R}}} + N_\mathbf{I}) \times d_{\text{model}}} \end{aligned} \tag{4}$$

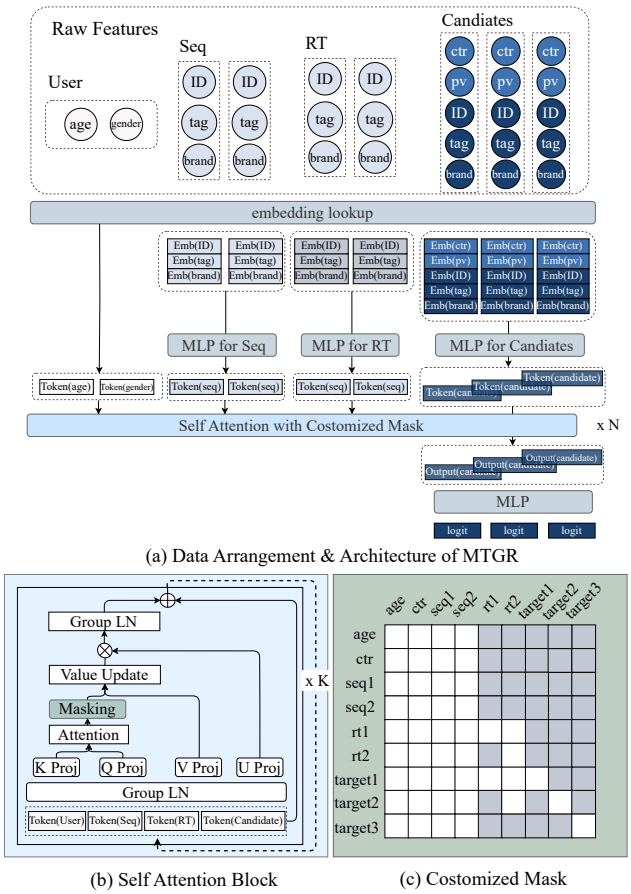

Figure 2: Data Arrangement and architecture of MTGR. (a) represents data arrangement and overall workflow of MTGR: 3 candidates' features are aggregated with the counterpart of one user. The features are embedded and converted to token by MLP, forming a series of input sequence for self-attention with mask. The representation of the tokens of candidates are used for logit via another MLP module. (b) Detailed description of self-attention module of L layers: input sequence is firstly processed by a group layer norm and 4 projections (Q/K/V/U). Self-attention is conducted with a customized mask. After value update, a dot-producted is performed on updated V and projected U. Finally, the updated value is again group-wise layer normed and added with a residual connection. (c) Customized mask to avoid information leakage, 'rt's and 'target's are ordered from recent to past: user feature ($\mathbf{U}, \overrightarrow{\mathbf{S}}$) is visible to all the tokens; 'rt's is visible to other tokens appear later; 'target's is visible to itself only.

4.2 Unified HSTU Encoder

Samples from a user are aggregated as a sequence of tokens, which is naturally suitable for processing using self-attention. Inspired by HSTU [22], MTGR uses stacks of self-attention layer and encoder-only architecture for modeling.

Similar in LLM, the input token sequences are processed layer-wisely. As shown in Fig.2, in the self-attention block, input sequence $\mathbf{X}$ is firstly normalized by group layer norm. Features of the same domain (for example, $\mathbf{U}$) form a group. Then, the group layer norm

ensures that tokens from different domains share a similar distribution before self-attention and aligns different semantic spaces of different domains $\tilde{X} = \text{GroupLN}(\mathbf{X})$. The normalized input is then projected to 4 different representation: $\mathbf{K}, \mathbf{Q}, \mathbf{V}, \mathbf{U} = \text{MLP}_{\mathbf{K/Q/V/U}}(\tilde{\mathbf{X}})$. $\mathbf{Q}, \mathbf{K}$ is utilized for multi-head attention computation with silu nonlinear activation. The leading attention is divided by total length of input features as an average factor. Next, a customized mask ($\mathbf{M}$) is imposed on the attention score and the projected $\mathbf{V}$ is applied for value update:

$$\tilde{\mathbf{V}} = \frac{\text{silu}(\mathbf{K}^T\mathbf{Q})}{(N_{\mathbf{U}} + N_{\overrightarrow{\mathbf{S}}} + N_{\overrightarrow{\mathbf{R}}} + N_{\mathbf{I}})}\mathbf{M}\mathbf{V} \tag{5}$$

The projected $\mathbf{U}$ and the updated $\tilde{\mathbf{V}}$ is dot-producted. Then, another group layer norm is applied. Finally, we add a residual module and place another MLP above it:

$$\mathbf{X} = \text{MLP}(\text{GroupLN}(\tilde{\mathbf{V}} \odot \mathbf{U})) + \mathbf{X} \tag{6}$$

Dynamic Masking [22] utilizes the causal mask for sequence modeling. However, such a implementation does not bring in significant improvement in MTGR. Besides, since $\overrightarrow{\mathbf{R}}$ records the user's most recent interactions which timing might coincide with the sample aggregation window. Using a simple causal mask in MTGR could result in information leakage. For example, interactions in the evening should not be exposed to candidates in the afternoon, yet this information may be aggregated into one sample. This dilemma requires a flexible and efficient masking.

In MTGR, $\mathbf{U}$, $\overrightarrow{\mathbf{S}}$ is regarded as static (in the following, we denote $\mathbf{U}$, $\overrightarrow{\mathbf{S}}$ as 'static sequence') because its information comes from before the aggregation window, thus it does not cause causality errors. $\overrightarrow{\mathbf{R}}$ is dynamic as it progressively include user's interaction ('dynamic sequence' for $\overrightarrow{\mathbf{S}}$) in real-time. MTGR applies full attention for static sequence, auto-regressive with dynamic masking for $\overrightarrow{\mathbf{R}}$ and diagonal masking for inter-candidates. Specially, 3 rules are set for MTGR's masking:

- The static sequence is visible to all tokens.
- The visibility of the dynamic sequence adheres to causality, where each token is only visible to tokens that occur afterward, which include candidate tokens.
- Candidate tokens ($\mathbf{C}$, $\mathbf{I}$) is visible to itself only.

Fig.2 (c) demonstrates an example for the dynamic masking: 'age', 'ctr' represent feature token from $\mathbf{U}$; 'seq1', 'seq2' for $\overrightarrow{\mathbf{S}}$; 'rt1', 'rt2' for $\overrightarrow{\mathbf{R}}$ and 'target1' - 'target3' for candidates. White block in row represents that the certain token is able to use the information from others while column represents the token is visible to others or not. Full attention is utilized for $\mathbf{U}$ and $\mathbf{S}$, leading to a white square from 'age' to 'seq2'. For 'rt1' to 'rt2', we assume that 'rt1' appear latter than 'rt2', Therefore, a tiny square from 'rt1' to 'rt2' is constructed with white blocks in upper triangle, meaning that 'rt1' is able to use information from 'rt2' while 'rt1' is not visible to 'rt2'. Besides, 'target2' and 'target3' is assumed to appear earlier than 'rt1', leading to 'rt1' is not visible to them. 'rt2' appear earlier than all 'target1' and 'target2' while latter than 'target3'. Therefore, 'rt2' is not visible to 'target3', which appears earlier than all 'rt's, leading to its unable to use information from 'rt's.

5 Training System

To facilitate the design and development of MTGR model structure and to conveniently incorporate more features from the fast-growing LLM community, we decided not to continue with our previous TensorFlow-based training framework. Instead, we opted to reconstruct our training framework within the PyTorch ecosystem. Specifically, we expanded and optimized TorchRec's functionality, made enhancements specifically catering to MTGR model's characteristics, and ultimately accomplished efficient training of MTGR model. Compared to TorchRec, our optimized framework improves training throughput by 1.6x – 2.4x while achieving good scalability when running over 100 GPUs. Compared to the DLRM baseline, we achieved 65x FLOPs per sample for forward computation, while the training cost remained nearly unchanged. We provide some of our key work as follows.

Dynamic Hash Table. TorchRec use a fixed-size table to process sparse embeddings, which is not suitable for large-scale industrial streaming training scenarios. Firstly, additional embeddings cannot be allocated to new users and items at real-time once the static table hits its preset capacity. Although some default embeddings may be used in this case, or some eviction strategies can be used to eliminate 'old' IDs, model accuracy will be degraded. Secondly, static tables typically require pre-allocation of capacity exceeding actual requirements to prevent ID overflow, inevitably leading to inefficient memory utilization. To tackle the challenges, we implement a hash-based and high performance embedding table, dynamically allocating space for sparse IDs during the training process. We employ a decoupled architecture for hash table, which separates key and value storage into distinct structures. The key structure maintains a lightweight mapping table containing keys and corresponding pointers to embedding vectors, while the value structure stores both the embedding vectors and auxiliary metadata (e.g., counters and timestamps) required for eviction policies. This dual-structure design achieves two critical objectives: (1) enabling dynamic capacity expansion through replication of only the compact key structure rather than huge embedding data, and (2) optimizing key scanning efficiency by storing keys in compact memory layouts while accommodating potentially sparse key distributions.

Embedding Lookup. The embedding lookup process employs All-to-all communication for cross-device embedding exchange. We adopt two-stage ID unique operation to reduce redundant IDs before and after ID communication to avoid repetitive embedding transfers among the devices. We also design a feature configuration interface to implement automatic table merging, which can reduce the number of embedding lookup operators and thus speed up the process.

Load balance. In recommendation systems, user behavior sequences often exhibit a long-tail distribution where only a few users have long sequences, while most have short ones. This leads to significant computational load imbalance when training with a fixed batch size (abbreviated as BS). A common solution is to use sequence packing techniques[10], where multiple short sequences are merged into a longer one. However, this approach requires careful mask adjustments to prevent different sequences from interfering with each other during the attention calculation, which can be quite costly to implement. Our straightforward solution is to introduce

dynamic BS. Each GPU's local BS is adjusted based on the actual sequence length of the input data, ensuring a similar computational load. Additionally, we've tweaked the gradient aggregation strategy to weight each GPU's gradients according to its BS, maintaining computational logic consistency with fixed BS.

Other Optimizations. To further enhance training efficiency, we maximize parallelism through pipeline technology using three streams: *copy*, *dispatch*, and *compute*. Specifically, the copy stream loads input data from CPU to GPU, the dispatch stream performs table lookups based on IDs, and the compute stream handles forward computation and backward updates. For instance, while the compute stream executes forward and backward passes for batch **T**, the copy stream concurrently loads batch **T** + 1 to mask I/O latency. Upon completing backward updates for batch **T**, the dispatch stream immediately initiates table lookups and communication for batch **T** + 1. Moreover, we employ bf16 mixed-precision training and develop a well-crafted cutlass-based attention kernel which is similar to FlashAttention to speed up the training process.

6 Experiments

6.1 Experiment setup

Datasets. Public datasets widely use independent ID and attribute features, where cross features are seldom introduced. However, cross features demonstrate importance in real-world application. Cross features are an important category of features in our scenario. They are often meticulously hand-crafted, including interactions like user-item, user and higher-level categories, item and spatio-temporal information. To compensate the absence of cross feature in public datasets, we construct a training dataset based on logs from the real industrial-scale recommendation system in Meituan. Unlike public datasets, our real dataset contains a richer cross features set and longer user behavior sequences. In addition, the volume of our dataset is large, allowing complex models to achieve more adequate convergence during training. For the offline experiments, we collect data over a 10-day period. The statistics of the dataset are shown in Table 1. For the online experiments, in order to compare with the DLRM baseline that has been trained for over 2 years, we constructed a longer-term dataset for the experiments, using data spanning more than 6 months.

Table 1: Dataset Statistics

Dataset	#Users	#Items	#Exposure	#Click	#Purchases
Train	0.21 billion	4,302,391	23.74 billion	1.08 billion	0.18 billion
Test	3,021,198	3,141,997	76,855,608	4,545,386	769,534

Baseline. For DLRM, we compare two methods in sequence modeling: SIM based on sequence retrieval and End2End modeling for original long sequences (E2E). In terms of scaling, we compared DNN, MoE[13], Wukong[24], MultiEmbed[6], and UserTower.

MoE uses 4 experts, with each expert containing a network of the same complexity as the base DNN. Wukong and MultiEmbed are configured to the same computational complexity as MoE. UserTower uses a set of learnable queries and inserts a qFormer[11] layer with another MoE (16 experts) module on user behavior. UserTower's computational complexity is tripled than MoE method, but it can share this computation for multiple predicted items for the same user during inference, thereby reducing inference costs. It has achieved good results in our scenario.

MTGR employs E2E to handle all sequence information. Additionally, as shown in Table 2, we have set up three different scales to verify the scalability of MTGR.

Table 2: Comparison of models with different settings and computational complexity

Model	Setting	Learning rate	GFLOPs/example
UserTower-SIM	-	8×10^{-4}	0.86
MTGR-small	$n_{\text{layer}} = 3$, $d_{\text{model}} = 512$, $n_{\text{heads}} = 2$	3×10^{-4}	5.47
MTGR-medium	$n_{\text{layer}} = 5$, $d_{\text{model}} = 768$, $n_{\text{heads}} = 3$	3×10^{-4}	18.59
MTGR-large	$n_{\text{layer}} = 15$, $d_{\text{model}} = 768$, $n_{\text{heads}} = 3$	1×10^{-4}	55.76

Evaluation Metrics. In offline, we focus on learning of two tasks: CTR and CTCVR (Click-Through Conversion Rate) and use AUC[5] and GAUC (Group AUC) for evaluation. GAUC is averaged on the the AUCs under users. Compared to AUC, GAUC pays more attention to the model's ranking ability for the same user. For online evaluation, we focus on two metrics: PV_CTR (CTR per page view) and UV_CTCVR (CTCVR per user view), where UV_CTCVR is the most crucial metric for evaluating the growth of business.

Parameter Setting. Our model is trained using the Adam optimizer. For DLRM, each GPU processes a batch size of 2400, with 8 NVIDIA A100 GPUs for training. In the case of MTGR, the batch size is set to 96, utilizing 16 NVIDIA A100 GPUs for training. As shown in Table 2, the learning rate decreases as the complexity of the model increases. Moreover, as the model size grows, we proportionally increase the size of the sparse parameters. Assuming that a token consists of k features, the embedding dimension for each feature is generally set to an integer close to d_{model}/k. Finally, the maximum length of $\overrightarrow{\mathbf{S}}$ is set to 1000, while $\overrightarrow{\mathbf{R}}$ is set to 100.

6.2 Overall Performance Comparison

We evaluate the performance of MTGR and other baseline methods using our 10-day dataset. Table 3 shows the performance of different models. The differences among various models across different offline metrics are quite consistent. Based on previous experience, an increase 0.001 in our offline metric is considered significant. Among the various versions of DLRM, Wukong-SIM and MultiEmbed-SIM achieve a better result than MoE-SIM. UserTower-SIM performs the best and UserTower-E2E shows a slight decline in performance compared to UserTower-SIM. We speculate that under the DLRM paradigm, the model complexity is insufficient to model all sequence information, leading to underfitting. Our proposed MTGR, even the smallest version, exceeds the strongest DLRM model. Additionally, models of three different sizes exhibit scalability, with their performance smoothly increasing as the complexity of the model increases.

6.3 Ablation Study

We perform ablation studies on two components of MTGR: Dynamic Masking and group layer norm (GLN) based on our small one. The ablation results are shown in Fig 4. Eliminating any of these from MTGR will result in a significant decline in performance, with the degree of decline being comparable to the increase from MTGR-small to MTGR-medium. This indicates the importance of

Table 3: Overall performance. Impr.% represents the relative improvement of the best MTGR model compared to the strongest DLRM baselines (underlined).

Model	CTR		CTCVR	
	AUC	GAUC	AUC	GAUC
DNN-SIM	0.7432	0.6679	0.8737	0.6504
MoE-SIM	0.7484	0.6698	0.8750	0.6519
MultiEmbed-SIM	0.7501	0.6715	0.8766	0.6525
Wukong-SIM	0.7568	0.6759	0.8800	0.6530
UserTower-SIM	<u>0.7593</u>	<u>0.6792</u>	0.8815	<u>0.6550</u>
UserTower-E2E	0.7576	0.6787	<u>0.8818</u>	0.6548
MTGR-small	0.7631	0.6826	0.8840	0.6603
MTGR-medium	0.7645	0.6843	0.8849	0.6625
MTGR-large	**0.7661**	**0.6865**	**0.8862**	**0.6646**
Impr.%	**0.8956**	**1.0748**	**0.4990**	**1.4656**

Table 4: Ablation studies of MTGR

Model	CTR		CTCVR	
	AUC	GAUC	AUC	GAUC
MTGR-small	0.7631	0.6826	0.8840	0.6603
w/o cross features	0.7495	0.6689	0.8736	0.6514
w/o GLN	0.7606	0.6809	0.8826	0.6585
w/o dynamic mask	0.7620	0.6810	0.8828	0.6587

Dynamic Masking and GLN for MTGR. In addition, we conducted extra experiments on the importance of cross features to MTGR. After removing the cross features, the performance metrics show a significant drop, which even erases the gain of MTGR-large over DLRM, highlighting the vital role of cross features in the real recommendation system.

6.4 Scalability

Fig. 3 illustrates the scalability of our MTGR. We perform tests based on MTGR-small for three different hyperparameters: the number of HSTU blocks, the d_{model}, and the input sequence length. As can be observed, MTGR demonstrates good scalability across different hyperparameters. Furthermore, Fig. 3(d) presents a power-law relationship between performance and computational complexity. The vertical axis indicates the gain in the CTCVR GAUC metric relative to our best DLRM model, UserTower-SIM, while the horizontal axis reflects the logarithmic multiple of computational complexity compared to UserTower-SIM.

6.5 Online Experiments

To further validate the effectiveness of MTGR, we deployed MTGR in the Meituan Take-away platform, conducting AB testing with 2% of the traffic. The volume of experimental traffic covers millions of exposures per day, demonstrating the confidence of the experiment. The comparison baseline is the most advanced online DLRM model (UserTower-SIM), and it has been continuously learning for 2 years. We use data from the last 6 months to train the MTGR model, which is then deployed online for comparison.

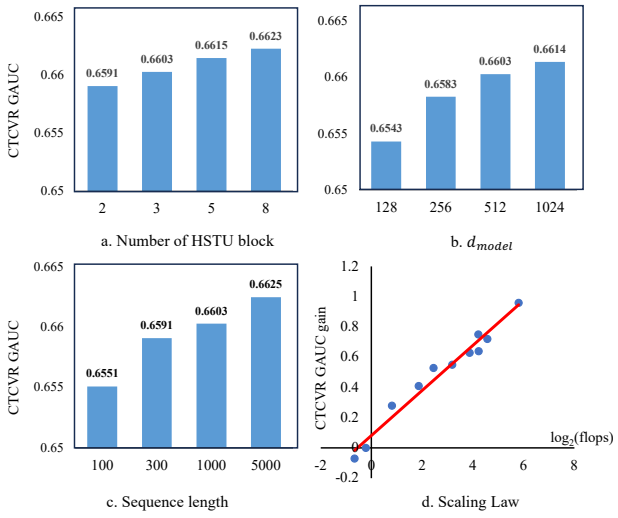

Figure 3: MTGR performance improves smoothly as we increase the numer of HSTU block, d_{model} and sequence length for training

Although the volume of training data is significantly lower than the counterpart of DLRM model, the offline and online metrics still greatly exceed the DLRM base. As shown in Table 5, both the offline and online metrics demonstrate scalability. We also found that as the number of training tokens increases, the benefits compared to DLRM continue to amplify. Eventually, in terms of CTCVR GAUC, our large version even exceeds the cumulative increase of all optimizations over the past year.

The model has already been fully deployed in our scenario, with training costs equal to DLRM, and inference costs reduced by 12%. For DLRM, its inference cost is approximately linear with the number of candidates. However, MTGR uses user aggregation for all candidates in a request, leading to sub-linear inference cost scaling with the number of candidates. This helps us reduce the overhead of online inference.

Table 5: Comparison of offline and online effects for different versions of MTGR.

	Offline Metric diff		Online Metric diff	
	CTR GAUC	CTCVR GAUC	PV_CTR	UV_CTCVR
MTGR-small	+0.0036	+0.0154	+1.04%	+0.04%
MTGR-medium	+0.0071	+0.0182	+2.29%	+0.62%
MTGR-large	+0.0153	+0.0288	+1.90%	+1.02%

7 Conclusion

In this paper, we proposed MTGR, a new ranking framework to explore the scaling law in recommendation systems based on HSTU. MTGR combines the advantages of DLRM and GRM, allowing the use of cross-features to ensure model performance while having the same scalability as GRM. MTGR has already been deployed in our scenario and has yielded significant benefits. In the future, we will explore how to extend MTGR to multi-scenario modeling, similar to large language models, to establish a recommendation foundation model with extensive knowledge.

References

[1] Qiwei Chen, Changhua Pei, Shanshan Lv, Chao Li, Junfeng Ge, and Wenwu Ou. 2021. End-to-end user behavior retrieval in click-through rateprediction model. *arXiv preprint arXiv:2108.04468* (2021).

[2] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[3] Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. 2025. OneRec: Unifying Retrieve and Rank with Generative Recommender and Iterative Preference Alignment. *arXiv preprint arXiv:2502.18965* (2025).

[4] Yan Fang, Jingtao Zhan, Qingyao Ai, Jiaxin Mao, Weihang Su, Jia Chen, and Yiqun Liu. 2024. Scaling laws for dense retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1339–1349.

[5] Cesar Ferri, José Hernández-Orallo, and Peter A Flach. 2011. A coherent interpretation of AUC as a measure of aggregated classification performance. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 657–664.

[6] Xingzhuo Guo, Junwei Pan, Ximei Wang, Baixu Chen, Jie Jiang, and Mingsheng Long. 2023. On the embedding collapse when scaling up recommendation models. *arXiv preprint arXiv:2310.04400* (2023).

[7] Ruidong Han, Qianzhong Li, He Jiang, Rui Li, Yurou Zhao, Xiang Li, and Wei Lin. 2024. Enhancing CTR Prediction through Sequential Recommendation Pretraining: Introducing the SRP4CTR Framework. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 3777–3781.

[8] Dmytro Ivchenko, Dennis Van Der Staay, Colin Taylor, Xing Liu, Will Feng, Rahul Kindi, Anirudh Sudarshan, and Shahin Sefati. 2022. Torchrec: a pytorch domain library for recommendation systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 482–483.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[10] Mario Michael Krell, Matej Kosec, Sergio P Perez, and Andrew Fitzgibbon. 2021. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *arXiv preprint arXiv:2107.02027* (2021).

[11] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*. PMLR, 19730–19742.

[12] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1754–1763.

[13] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1930–1939.

[14] William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*. 4195–4205.

[15] 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.

[16] Kyuyong Shin, Hanock Kwak, Su Young Kim, Max Nihlén Ramström, Jisu Jeong, Jung-Woo Ha, and Kyung-Min Kim. 2023. Scaling law for recommendation models: Towards general-purpose user representations. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 4596–4604.

[17] Zihua Si, Lin Guan, ZhongXiang Sun, Xiaoxue Zang, Jing Lu, Yiqun Hui, Xingchao Cao, Zeyu Yang, Yichen Zheng, Dewei Leng, et al. 2024. Twin v2: Scaling ultra-long user behavior sequence modeling for enhanced ctr prediction at kuaishou. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4890–4897.

[18] Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. 2020. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM conference on recommender systems*. 269–278.

[19] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*. 1785–1797.

[20] Xu Wang, Jiangxia Cao, Zhiyi Fu, Kun Gai, and Guorui Zhou. 2024. HoME: Hierarchy of Multi-Gate Experts for Multi-Task Learning at Kuaishou. *arXiv preprint arXiv:2408.05430* (2024).

[21] Bencheng Yan, Shilei Liu, Zhiyuan Zeng, Zihao Wang, Yizhen Zhang, Yujin Yuan, Langming Liu, Jiaqi Liu, Di Wang, Wenbo Su, et al. 2025. Unlocking Scaling Law in Industrial Recommendation Systems with a Three-step Paradigm based Large User Model. *arXiv preprint arXiv:2502.08309* (2025).

[22] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152* (2024).

[23] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 12104–12113.

[24] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. 2024. Wukong: Towards a scaling law for large-scale recommendation. *arXiv preprint arXiv:2403.02545* (2024).

[25] Wei Zhang, Dai Li, Chen Liang, Fang Zhou, Zhongke Zhang, Xuewei Wang, Ru Li, Yi Zhou, Yaning Huang, Dong Liang, et al. 2024. Scaling User Modeling: Large-scale Online User Representations for Ads Personalization in Meta. In *Companion Proceedings of the ACM Web Conference 2024*. 47–55.

[26] 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.