# Generalized Deep Mixed Models

Jun Shi, Chengming Jiang, Aman Gupta, Mingzhou Zhou, Yunbo Ouyang
Charles Xiao, Qingquan Song, Alice Wu, Haichao Wei, Huiji Gao
LinkedIn Corporation
Mountain View, CA, USA
{jshi,cjiang,amagupta,mizhou,youyang,qxiao,qsong,aiwu,hawei,hgao}@linkedin.com

## ABSTRACT

We introduce generalized deep mixed model (GDMix), a class of machine learning models that combines the power of deep and linear models for large-scale response prediction for recommender systems. GDMix leverages deep neural networks (DNNs) or equivalently powerful global models (fixed effects), and augments their performance by learning entity-specific personalized models (random effects). For instance, the click response from a particular user $m$ to a job posting $j$ may consist of contributions from a DNN model common to all job postings and users, a model specific to $m$ and a model specific to $j$. In this paper, we use DNNs for fixed effects and logistic regression for random effects. This class of models not only possesses powerful modeling capabilities, but also can be trained and updated efficiently for very large scale recommender systems. GDMix-based models now power major recommender systems at LinkedIn, and we demonstrate their powerful capabilities by reporting significant increase in relevance and engagement for Feed and Ads recommendation at LinkedIn. The source code for the GDMix training framework is available at https://github.com/linkedin/gdmix under the BSD-2-Clause License.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

## KEYWORDS

generalized deep mixed model, neural networks, recommender systems, response prediction

**ACM Reference Format:**
Jun Shi, Chengming Jiang, Aman Gupta, Mingzhou Zhou, Yunbo Ouyang, Charles Xiao, Qingquan Song, Alice Wu, Haichao Wei, Huiji Gao. 2022. Generalized Deep Mixed Models. In *Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456


*KDD '22, August 14-18, 2022, Washington, DC*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Social networks provide rich content to users, including news articles, blog posts, job postings, advertisements and online courses. Personalization, targeting relevant items to users, is crucial for improving user experience and increasing user engagement.

Machine learning models are widely used to predict a user's affinity to an item, hence proving useful for recommendations. These models are expected to tackle two aspects of the personalized recommendation problem - generalization and memorization. This allows such models to recommend unseen or existing items to current or new users.

In a deep neural network (DNN)-based model, generalization can be modeled by generic features that are not tied to a single user or item and memorization can be captured by personalized features such as user or item ID embeddings. Take jobs recommendation as an example. If we consider job descriptions and user profiles as sources of features, the resultant features will be derived mostly from text data. For instance, jobs and users can be represented by aggregating word embeddings. To further enrich the features, one can add user and job IDs which in turn can be represented by ID-based dense embeddings. The ID-based features are of the finest granularity as far as personalization is concerned. Including all these features in a single model can simplify the training and deployment process but is hard to scale, especially when the number of IDs is very large. In the above example, the cardinality of user ID may be in the order of 100 million, and hence the corresponding embedding table can become unmanageably large. Furthermore, if cross features such as user ID cross job title are used, the table may swell to 10 billion or more rows. Such large models are difficult to train efficiently.

GDMix, following the principles of mixed-effects modeling frameworks such as GLMix [21], decouples the generalization and memorization tasks. Similar to GLMix, the GDMix fixed effects model, a.k.a. the global model, captures the general response from the entire training dataset. It can be applied to unseen users or items as long as they can be featurized. Unlike GLMix, GDMix allows the fixed effects model to be a powerful DNN model or a similarly sophisticated model like gradient boosted decision trees (GBDTs) [6]. The random effects models are ID-specific and capture idiosyncrasies of users and items. The overall response is the sum of the responses from all effects. Let's consider an example - we would like to recommend jobs to Alice, who is a software engineer. The global model may give a high score for "software engineer" and "machine learning", but Alice's own browsing history may suggest that she prefers "database" to "machine learning", which is captured in Alice's random effects model. The overall response may rank "database" jobs higher than "machine learning" jobs due to the contribution of Alice's own model output.

This decoupling makes the models more scalable since the random effects are trained after the fixed effects, and the individual random effects models can be trained independently in parallel. It makes fast (sub-hour) update of models with tens to hundreds of millions of ID-based embeddings possible, a scenario crucial to the success of freshness-sensitive applications like Feed recommendation and Ads click-through-rate (CTR) prediction. We provide more evidence in the experiments section.

### 1.1 Related Work

Use of machine learning models for personalized recommender systems is an active area of research. The bulk of the literature is focused on single deep learning models. Early work includes wide-and-deep [4] models, where a wide linear model (for memorization) is used alongside a DNN (for generalization). To include more feature interactions, DeepFM [10] replaces the logistic regression in the wide-and-deep model with factorization machines [17]. Deep Interest Network [23] applies a local activation unit to a set of item ID features to identify a user's interest. In Deep Interest Evolution Network [22], GRU units and attention mechanism are used to analyze user behavior sequence. Deep Learning Recommendation Model [14] is a general framework where a top multilayer perceptron (MLP) and a bottom MLP are used to capture interactions between different ID embeddings. A common problem with the single model approach is that as the cardinality of ID features increases, the models become too big to be trained and served efficiently. Model reduction techniques such as mixed dimension embeddings [8], compositional embedding [19] have been proposed to enable better scaling.

Earlier work along the line of mixed-effect modeling includes GLMix [21] where the fixed effects and random effects are all logistic regression models. Large metric lifts prove the effectiveness of the random effects. Near-realtime training frameworks like Lambda Learner [16] demonstrate the critical need for high model update frequency for Ads CTR prediction and focus on fast refresh of the random effects coefficients using the GLMix framework.

### 1.2 Our Contributions

In this paper, we present GDMix, a deep learning-based personalization model training framework for large scale recommender systems and detail its success across LinkedIn applications. GDMix combines the powerful modeling capabilities of DNNs or GBDTs and the agility of linear models like logistic regression. It improves upon GLMix [21] with the following enhancements: (1) Support for sophisticated fixed effect models in the form of arbitrary DNNs and other non-linear models such as GBDTs. (2) Efficient training mechanisms for large-scale models consisting of a large fixed effect model with potentially billions of parameters, and hundreds of millions of smaller random effect logistic regression models. GDMix has now been widely adopted at LinkedIn, supplanting GLMix and powering the most critical recommendation systems at the company. Offline and online experiments demonstrate that the GDMix framework trains more performant models while consuming fewer compute resources, when compared to GLMix. GDMix is especially suited for freshness-sensitive use cases where very frequent model updates are required.

## 2 GDMIX MODELS

GDMix has its roots in the mixed effects model, a statistical model containing both fixed effects and random effects [2]. The definition of these effects varies for different applications. Five different definitions can be found in [7]. Instead of attempting to add another definition, we use an example to illustrate the idea. Assume we want to estimate the academic scores for students in a city. Possible features are gender, household income, parents' education levels. These are applicable to all students, thus can be treated as fixed effects. Time spent studying is another feature. However, it has a varying impact on a student's test scores. Some students master time management and are able to score better per unit of time spent studying. Others may get less return from time spent studying. We can model time spent studying as a per-student random effect. The estimated model score for each student is the sum of model scores from the fixed effects and random effects.

We discuss GDMix with emphasis on recommender systems in this section. We start with mathematical definitions in Section 2.1, then move on to efficient training strategies in Section 2.2. Finally, we demonstrate the application of GDMix models at LinkedIn in Section 2.3.

### 2.1 Theory

Since we use GDMix mostly to solve various click-through-rate prediction problems at LinkedIn, our focus will be on binary classification. Let's consider a single training example for simplicity. Assume the estimated binary response $y$ follows the Bernoulli distribution with parameter $p$, i.e. the expectation $\mathcal{E}(y) = p$. The logit function $g(p)$ (also known as the link function for the Bernoulli distribution in the context of generalized linear models) is defined as $\log(p/(1-p))$. Then we have the following:

$$g(p) = f(\mathbf{x}_f, \mathbf{w}_f) + \sum_i r_i(\mathbf{x}_{ri}, \mathbf{w}_{ri}), \tag{1}$$

where functions $f(\cdot)$ and $r_i(\cdot)$ are the fixed effects and the $i$-th random effects model, respectively. $\mathbf{x}_f$ and $\mathbf{w}_f$ are the features and weights for the fixed effects model. $\mathbf{x}_{ri}$ and $\mathbf{w}_{ri}$ are the features and weights for the $i$-th random effects model. Note that GDMix allows for more than one random effects model. For example, the LinkedIn learning course recommendation model includes a per-user model (where we train a model for each user) and a per-course model (where we train a model for each course). In the case of GLMix, both $f(\cdot)$ and $r_i(\cdot)$ are linear functions, i.e. both fixed effects model and random effects models are logistic regression models. In theory, a GDMix model allows both to be arbitrary non-linear functions. In practice, we use non-linear functions (deep neural networks, gradient boosting decision trees [6], etc.) for $f(\cdot)$ while using linear functions for $r_i(\cdot)$ due to the following considerations. (1) Each random effects model may have too few training examples to support most nonlinear models. For instance, a per-user model is trained on a user's logged activities which may be in the order of a hundred per month, insufficient to train a neural network. (2) We have a large number of random effects models, sometimes in the order of a hundred million. It is difficult to train so many nonlinear models efficiently.

During the training stage, traditional Bayesian methods compute the posterior probability distribution of the weights given the

training dataset. Then during inference, the posterior probability of the weights is integrated out to arrive at the probability of a label. Both steps incur prohibitively large computational load. Instead, we choose to do a point estimate by maximizing the log posterior probability of the weights, given the training dataset.

$$\begin{aligned}
&\max_{\mathbf{w}_f,\{\mathbf{w}_{rij}\}} \log \Pr(\mathbf{w}_f, \{\mathbf{w}_{rij}\} | \mathbf{x}_{fn}, \{\mathbf{x}_{rin}\}, y_n, \forall n \in \mathcal{T}) \\
\propto &\max_{\mathbf{w}_f,\{\mathbf{w}_{rij}\}} \sum_{n\in\mathcal{T}} \log \Pr(\mathbf{x}_{fn}, \{\mathbf{x}_{rin}\}, y_n | \mathbf{w}_f, \{\mathbf{w}_{rij}\}) \\
&+ \log \Pr(\mathbf{w}_f) + \sum_i \log \Pr(\mathbf{w}_{rij}) \\
= &\max_{\mathbf{w}_f,\{\mathbf{w}_{rij}\}} \sum_{n\in\mathcal{T}} \Big( y_n \log \sigma(s_n(\mathbf{w}_f, \{\mathbf{w}_{ri\mathbf{I}(i,n)}\})) \\
&+ (1-y_n) \log(1-\sigma(s_n(\mathbf{w}_f, \{\mathbf{w}_{ri\mathbf{I}(i,n)}\})))\Big) \\
&+ \log \Pr(\mathbf{w}_f) + \sum_i \log \Pr(\mathbf{w}_{rij}).
\end{aligned} \tag{2}$$

The vector $\mathbf{w}_f$ is the weights for the fixed effects model. $\{\mathbf{w}_{rij}\}$ is the set of weights for the random effects models, where subscript $i$ is over all random effects models and subscript $j$ is over all IDs belonging to a specific random effects model. $\mathcal{T}$ is the training dataset. $\mathbf{x}_{fn}$ represents the fixed effects features for the $n$-th training example. $\mathbf{x}_{rin}$ is the $i$-th random effect features from $n$-th example. $\mathbf{I}(i,n)$ is the index function that extracts the entity ID for the $i$-th random effects model from the $n$-th example. $\Pr(\mathbf{w}_f)$ and $\Pr(\{\mathbf{w}_{rij}\})$ are the prior distribution of $\mathbf{w}_f$ and $\{\mathbf{w}_{rij}\}$, respectively. $\sigma(\cdot)$ is the sigmoid function. $s_n(\mathbf{w}_f, \{\mathbf{w}_{rij}\})$ is the logit for the $n$-th example as defined below.

$$s_n(\mathbf{w}_f, \{\mathbf{w}_{rij}\}) = f(\mathbf{x}_{fn}, \mathbf{w}_f) + \sum_i r_i(\mathbf{x}_{rin}, \mathbf{w}_{ri\mathbf{I}(n)}) \tag{3}$$

It is worth noting that the fixed effects and random effects models can be trained separately. A common usage at LinkedIn is as follows. The fixed effects model is trained first, either within the GDMix training framework or by some other dedicated trainer, e.g. a gradient boosting decision tree trainer. This training happens less frequently, once a week or even less frequently than that. The scores from the fixed effects model are provided as offsets to the random effects models, which can be trained as frequently as once every two hours. During random effects model training, Eq.1 simplifies as follows for the $n$-th sample.

$$g(p) = \text{offset}_{fn} + \sum_i r_i(\mathbf{x}_{rin}, \mathbf{w}_{ri\mathbf{I}(n)}), \tag{4}$$

where $\text{offset}_{fn}$ is the output logit from the fixed effects model for the $n$-th example.

## 2.2 Model Training

Before we dive into the methodology of training GDMix models, let us examine the size of the models. The fixed effects model weights $\mathbf{w}_f$ can range from a few thousand to billions of parameters. A single random effects model $\mathbf{w}_{rij}$ can be of ten to a thousand dimension. The index $i$ specifying the random effects models usually range from 0 to 2 (for example, each index representing a random effects model tied to a different entity type: job, member, etc.). The range of index $j$ representing the entity ID can reach as large as $10^8$. Thus the total number of parameters in the model can reach $10^{11}$. Models of this size are difficult to train efficiently.

Luckily, the additive nature of Eq.3 provides an efficient solution. First, the fixed effects and random effects models can be trained sequentially. For example, we can fix $\{\mathbf{w}_{rij}\}$ and update $\mathbf{w}_f$. Then we update $\mathbf{w}_{r0j}$ while fixing $\mathbf{w}_f$ and $\{\mathbf{w}_{rij}, i \neq 0\}$. This block-coordinate-descent method allows efficient training of individual effects model. For each random effects model, further division is possible. Let's assume we plan to train a per-user random effects models which consists of millions of user IDs. Since the training data for each user don't overlap, we can partition the entire dataset by user IDs and train the per-user models in parallel. This divide-and-conquer approach results in high training efficiency. In some cases, it only take a few minutes to train random effects models with millions of IDs.

For the non-linear fixed effects models, common algorithms from the stochastic gradient descent (SGD) family are often used to find a local optimum. For logistic regression models, Quasi-Newton methods such as L-BFGS [3] are preferred due to fast convergence. As a result, we support a mixture of the solvers in GDMix, for example, Adam [13] for the fixed effects deep neural networks and L-BFGS for random effects logistic regression models. This combination is the key to efficient training and better model quality.

## 2.3 GDMix at LinkedIn

LinkedIn serves millions of users every day by providing relevant recommendations for a variety of applications. The GDMix framework is a horizontal system that any LinkedIn business unit can simply plug in to its pipeline. We introduce three use cases in this section.

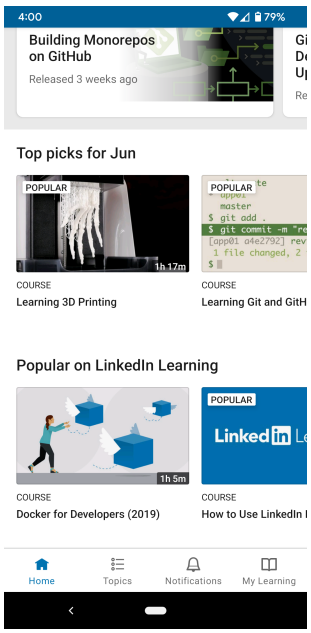



**Figure 1: A screenshot of the LinkedIn Learning application**

*2.3.1 Course Recommendation.* LinkedIn Learning provides many online courses to help users learn business, creative and technological skills to achieve their personal and professional goals. An example of the LinkedIn Learning page is shown in Figure 1. Recommended courses are based on users' past interaction with LinkedIn content, semantic similarity between user profiles and course description, among other features. User engagement (including user sign up and engagement with course content) is the true north business metric, while click-through-rate is a proxy indicator we use to optimize the GDMix model. The overall model includes a fixed effects model, a per-user random effects model and a per-course random effects model. The logit for the $n$-th example is:

$$\begin{aligned} s_n(\mathbf{w}_f, \mathbf{w}_{c\mathbf{I}(c,n)}, \mathbf{w}_{m\mathbf{I}(m,n)}) &= f(\mathbf{x}_{fn}, \mathbf{w}_f) + r_c(\mathbf{x}_{cn}, \mathbf{w}_{c\mathbf{I}(c,n)}) \\ &\quad + r_m(\mathbf{x}_{mn}, \mathbf{w}_{m\mathbf{I}(m,n)}) \\ &= \mathbf{x}_{fn}^\top \mathbf{w}_f + \mathbf{x}_{cn}^\top \mathbf{w}_{c\mathbf{I}(c,n)} \\ &\quad + \mathbf{x}_{mn}^\top \mathbf{w}_{m\mathbf{I}(m,n)}, \end{aligned} \tag{5}$$

where $\mathbf{w}_f$, $\mathbf{w}_{c\mathbf{I}(c,n)}$ and $\mathbf{w}_{m\mathbf{I}(m,n)}$ are the weights for the fixed effects, per-course random effects and per-user random effects models, respectively. $\mathbf{x}_{fn}$, $\mathbf{x}_{cn}$ and $\mathbf{x}_{mn}$ are the corresponding feature vectors. $\mathbf{I}(c,n)$ specifies the per-course random effects model index for the $n$-th example. $\mathbf{I}(m,n)$ specifies the per-user random effects model index for the $n$-th example. Since all effects are modeled as logistic regression, L-BFGS is used to find the optimal weights.

The fixed effects model is relatively stable, hence it is only trained once in a while. The random effects are trained a few times a week, during which the fixed effects model is locked and used for inference only.

*2.3.2 Advertising.* This is a multi-billion dollar business at LinkedIn and its success is closely related to the company's bottom line. Our goal is to provide highly relevant Ads to the users so that they are interested in engagement. Since click-through-rate (CTR) is an important relevance metric, whether a user clicks on an Ad or not is tracked and used as the training label. The GDMix model consists of a fixed effects model and a per-advertiser random effects model. The overall logit for the $n$-th example is the sum of the two model outputs.

$$s_n(\mathbf{w}_f, \mathbf{w}_{a\mathbf{I}(a,n)}) = f(\mathbf{x}_{fn}, \mathbf{w}_f) + \mathbf{x}_{an}^\top \mathbf{w}_{a\mathbf{I}(a,n)}. \tag{6}$$

The fixed effects model is a deep neural network with features from user activities, user devices, ads content, etc. The random effects model is a logistic regression model using all ID features related to ads campaign, ads channel, etc. Since model freshness is crucial to Ads, training is split into two phases. In the first phase (the cold start phase), the entire GDMix model is trained from scratch once every few weeks. Then in the second phase (the warm start phase) the fixed effects model is locked while the random effects model is trained once every few hours. Increased frequency of random effects updates has been shown to improve model performance [16].

*2.3.3 Feed.* Feed is the landing page when users access LinkedIn. The ranking of feed items has a very strong impact on user experience and satisfaction. A user can interact with a Feed item in various ways, e.g. click, like, comment, skip, etc. Thus, the Feed ranking model leverages multi-task learning (MTL) and multi-objective optimization (MOO). The MTL and MOO problem can be formulated as the fixed effects model, but the current version of GDMix is not capable of solving these kind of problems. It is currently solved by a dedicated training framework external to GDMix. The logit from the above model is then used as the offset to the GDMix model. The total logit for the $n$-th example can be formulated as

$$s_n(\mathbf{w}_f, \mathbf{w}_{o\mathbf{I}(o,n)}) = \text{offset}_n + \mathbf{x}_{fn}^\top \mathbf{w}_f + \mathbf{x}_{on}^\top \mathbf{w}_{o\mathbf{I}(o,n)}, \tag{7}$$

where $\text{offset}_n$ is the logit from the MTL-MOO model for the $n$-th example. $\mathbf{w}_f$ is the fixed effects model weight vector and $\mathbf{x}_{fn}$ is the fixed effects feature vector from the $n$-th example. The random effects models are indexed by the root object IDs, essentially all the items on the Feed homepage, e.g. a news article, a blog post, a job posting, etc. $\mathbf{x}_{on}$ and $\mathbf{w}_{o\mathbf{I}(o,n)}$ are the features and weights for the root object ID in the $n$-th training example.

Since the root objects have very short life cycle, e.g. a news article may become viral in a few hours, then fades away in another few hours, frequent training of the random effects model is critical to keep the recommendations relevant. The models are currently trained a few times a day, influenced by business requirements.

## 3 IMPLEMENTATION

The scale of the optimization problem in Eq.2 can be fairly large. There may be several billion examples in the training dataset. The fixed effects model may have a billion parameters. Random effects may consist of a hundred million logistic regression models. Often, there are stringent requirements for the time required to train. For example, the Ads warm start flow needs to update a hundred thousand per-advertiser models within a couple of hours. We discuss how GDMix is designed and implemented to meet these requirements.

### 3.1 Training Component

GDMix should support deep learning model training natively. Two possible routes are available, either using existing deep learning framework such as Tensorflow [1] and PyTorch [15] or building a new framework from scratch. Given the complexity of deep learning model training, it is desirable to select an existing framework as our base trainer. This decision has its own shortcomings. For example, off-the-shelf distributed training from Tensorflow or PyTorch is not efficient. The native SGD-like optimizers are slow to converge for logistic regression models. Despite these drawbacks it is still considerably easier to leverage them than to build our own deep learning trainer. In the end, we decided to build GDMix based on Tensorflow with a few heavily customized operations.

### 3.2 Data Processing Component

Preparing training data in the format that Tensorflow can consume is both absolutely necessary and non-trivial. The most efficient data format Tensorflow uses is TFRecord, which however is not a commonly used format in the ETL (extract, transform, and load) process. At LinkedIn, we use the Avro format for internal data storage. We leverage a batch conversion job for this purpose. In addition, the random effects models are trained on examples belonging to a single entity (e.g. a member or a job). Thus the training dataset has

to be grouped by the entity ID, then partitioned accordingly for parallel training.

Apache Spark is used for all data processing tasks. We developed and open-sourced a library called Spark-TFRecord for data format conversion. The library is designed as a Spark data source which we can use to convert most Spark supported data formats, such as Avro and Parquet, into the TFRecord format efficiently. It also supports the *partitionBy* operation, an essential step to generate training data for random effects models.

## 3.3 Block Coordinate Descent

Eq. 2 can have as many as a hundred billion parameters. It is difficult if not impossible to train them jointly in an efficient manner. We use a block coordinate descent method instead, i.e. one "effects" model is updated while the rest is fixed. This is an extension of the method used in GLMix [21]. Specifically, we use the following to update the fixed effects model parameters.

$$\begin{aligned}\mathbf{w}_f = \arg\max_{\mathbf{w}_f} \sum_{n\in\mathcal{T}} \Big(& y_n \log\sigma(s_n - f(\mathbf{x}_{fn}, \mathbf{w}_f^{\text{old}}) + f(\mathbf{x}_{fn}, \mathbf{w}_f))) \\ &+(1-y_n)\log(1-\sigma(s_n - f(\mathbf{x}_{fn}, \mathbf{w}_f^{\text{old}}) + f(\mathbf{x}_{fn}, \mathbf{w}_f)))\Big) \\ &+ \log\Pr(\mathbf{w}_f),\end{aligned} \tag{8}$$

where $s_n$ is the total logit from the last iteration. Once $\mathbf{w}_f$ is updated from Eq.8, $s_n$ is updated according to the following.

$$s_n^{\text{new}} = s_n^{\text{old}} - f(\mathbf{x}_{fn}, \mathbf{w}_f^{\text{old}}) + f(\mathbf{x}_{fn}, \mathbf{w}_f^{\text{new}}) \tag{9}$$

When it comes to the random effects models, we use the following rule to update the coefficients corresponding to $j$-th Id inside the $i$-th random effects model.

$$\begin{aligned}\mathbf{w}_{rij} = \arg\max_{\mathbf{w}_{rij}} \sum_{n|\mathbf{I}(n)=j} \Big(& y_n \log\sigma(s_n - \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}^{\text{old}} + \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}) \\ &+(1-y_n)\log(1-\sigma(s_n - \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}^{\text{old}} + \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}))\Big) \\ &+ \log\Pr(\mathbf{w}_{rkj}),\end{aligned} \tag{10}$$

where $s_n$ is the total logit given the last model parameters. Once $\mathbf{w}_{rij}$ is updated from Eq.10, $s_n$ is updated according to the following.

$$s_n^{\text{new}} = s_n^{\text{old}} - \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}^{\text{old}} + \mathbf{x}_{rn}^\intercal \mathbf{w}_{rij}^{\text{new}} \tag{11}$$

The entire training procedure is listed in Algorithm 1.

**Algorithm 1** Block coordinate descent for GDMix

```
while convergence condition not met do
    Update fixed effects model according to Eq.8
    Update s_n according to Eq. 9
    for i = 1, 2, ..., N_r do
        for all IDs in random effects i do in parallel
            Update random effects models according to Eq.10
            Update s_n according to Eq. 11
        end for
    end for
end while
```

## 3.4 Training Fixed Effects Model

GDMix supports training deep learning and logistic regression models natively. On the surface, the latter is a special case of the former. However, the requirements on the training performance are dramatically different. Deep neural networks do well with SGD-like optimizers, while quasi-Newton methods result in much faster convergence on the logistic regression problem. In order to match the training speed of GLMix, which uses quasi-Newton methods for all effects, we chose to treat these two cases differently.

The DNNs are trained with either the standard Tensorflow Keras API or with an open source ranking framework called DeText [11], if text-based features are used. Distributed training is based on Tensorflow's MirroredStrategy or MultiWorkerMirroredStrategy. We are also working on the support of Horovod [18] for more efficient communication among workers. All off-the-shelf optimizers from Tensorflow are automatically supported. Users can even train a list-wise ranking model via the DeText interface. Model training is immediately followed by model inference on the training and validation datasets. The inferenced scores are used for training in the next stage.

The logistic regression fixed effects model trainer is heavily customized. The computation graph is built with a low-level, session-based Tensorflow graph. The L-BFGS solver from SciPy [20] is employed to achieve fast convergence. The communication between the solver and the graph is through the *feed_dict* interface. Prior to training, the dataset is evenly divided into many partitions. The number of partitions is a multiple of the number of workers. Training is done in a synchronous manner. Each worker reads examples from its partition, computes and accumulates the gradient. Tensorflow all-reduce operator is called to average the gradients among all the workers. Each worker then runs the L-BFGS algorithm locally and independently on the averaged gradient. Similar to the deep model, after the training an inference step is run to obtain the scores for all example in the training and validation datasets. The scores are used for the subsequent random effects model training.

## 3.5 Training Random Effects Models

There are two modes in training random effects: entity mode and cohort mode. Entity mode is used when there are a large number of individual models while each model has a small training dataset. Cohort mode is used when the number of individual models is small while each model has a large number of training examples. Entity mode is applicable to the per-member models where a model is trained for each member. Cohort mode is applicable to the case where members are partitioned by countries or geographic regions and a model is trained for each partition. In this paper we focus on entity mode since it is more commonly used within LinkedIn.

In the entity mode, a single worker trains many entity models sequentially. The entire dataset is grouped and partitioned by the entity IDs. The partitions are stored on HDFS and assigned to the workers. Each worker trains models for the entities in its assigned partitions independently. The training data for an entity are grouped as a single super-record (a list of all the related examples) inside a partition. During training ,a worker reads the super-record one by one to memory, then runs a L-BFGS solver on each super-record to obtain the trained parameters for the corresponding entity.

Training efficiency is achieved due to the following design. First, workers operate independently of each other and random-effect models can be trained asynchronously. Second, usage of a super-record allows for a dataset to be completely loaded into memory, which reduces I/O operations during L-BFGS iterations. If the training data for an entity is too large to fit in the memory, we can truncate the number of training examples to a reasonable size during the data pre-processing phase. The cohort mode is recommended if the number of training examples can not be truncated.

### 3.6 Flow Orchestration

A GDMix workflow consists of a mixture of jobs written in Python or Scala. Training jobs, both fixed effects and random effects, are written in Python, while the data processing jobs, including data conversion, data partitioning and score computation are written in Scala. They need to be connected by a flow orchestrator. We use Azkaban internally and use Kubeflow for our open source code. With the orchestrator, we can schedule the flow automatically on a daily basis. A flow with one fixed effects model and two random effects models is illustrated in Fig.2.

Either fixed effects model or random effect models can be optional. For example the fixed effects model could be a gradient boosting decision tree model which is trained elsewhere. The scores from the tree model will be included in the training data as offset to the random effects. It is also possible to lock a model and update the rest. For example, since the fixed effects model usually does not need to be updated as often as random effects, we often lock the fixed effects model in Figure 2 and use it for scoring while training the per-member and per-job models daily.

## 4 EXPERIMENTS FOR LINKEDIN PRODUCTION SYSTEMS

LinkedIn operates some of the largest recommendation systems in the world, serving millions of users every day. This poses multiple challenges related to model accuracy, freshness and training speed. In this section, we detail how GDMix tackles the aforementioned challenges and highlight offline and online production wins achieved for highly impactful recommendation systems at LinkedIn.

### 4.1 Offline Evaluation

*4.1.1 Advertising.* Our baseline Ads click-through-rate prediction model is a GLMix model. Features fall into five categories: contextual, advertisement, user, advertiser, ad-user interaction. All features are categorical features. The number of parameters for the fixed and random effects models are in the order of a million. The training dataset consists of a few billion samples.

We build two GDMix models for this problem. One uses logistic regression for both the fixed and the random effects (identical in structure to the GLMix model), and the other uses a deep neural network as its fixed effects model and logistic regression for the random effects.

The GDMix linear model achieves the same AUC as the baseline model but also consumes far fewer compute resources - 15% faster in training time and 43% lower consumption of memory. The main reason for the reduction can be attributed to our highly efficient implementation of the divide-and-conquer approach during random effects model training.

The GDMix deep model uses a multi-layer perceptron (MLP). All categorical features are mapped to embeddings of size 20, before being fed to the MLP as input. We use the Adam optimizer [13] to train the deep model for 2 epochs. The result is a 1.6% AUC-ROC improvement over the logistic regression-based fixed effects model. After adding the random effects model to the fixed effect, the overall AUC lift is 0.76%, which is considered highly significant for this problem.

*4.1.2 Feed.* As stated earlier, we use multi-task multi-objective optimization (MTL-MOO) for Feed recommendation. While the base model is trained outside of the GDMix framework, GDMix contributes to the personalization aspect of the overall model. In particular, the fixed effects model is very simple. It takes scores from the MTL-MOO problem as an offset and only has 2 features of its own - the intercept and a position feature. Thus, the fixed effects model is used to remove position bias of a Feed item. The per-item random effects models are also simple, with no more than 10 features. One challenge is the heterogeneity of the items (blogs, jobs, ads, network updates), each of which follows a different distribution. Another challenge is the large number of items, which can be as large as 100 million. Maintaining a high level of parallelism is the key to training the random effects models efficiently. We use 800 workers for GDMix training, where each worker sequentially optimizes about a hundred thousand logistic regression models.

The massive parallelism shows a large improvement over the baseline GLMix trainer. GDMix reduces end-to-end training time by 45% and also consumes 54% fewer compute resources. Similar to the Ads use case, we attribute this to our efficient divide-and-conquer approach that allows us to train a very large number of models in parallel. Since capturing virality of feed items is crucial to serving more relevant and fresh content to users, the shortened training time yields significant lifts in freshness metrics in online A/B tests.

*4.1.3 Job recommendation.* Million of users leverage LinkedIn JYMBII (Job You May Be Interested In) to find relevant jobs. We apply GDMix models to the JYMBII recommendation problem, where relevant job postings are recommended to users on the Feed or the Jobs home page. Recommendations are based on users' public profiles and their past activities on the LinkedIn platform. Business success is measured by short-term click and apply counts, as well as long-term confirmed hire counts. We use click-through-rate as a proxy for business metrics and optimize the same. The baseline model is a GLMix model and has a global fixed effects model, a per-user random effects model and a per-job random effects model (Figure 2). The features fall into three categories: user, job, and user-job interactions. The user ID has a cardinality in the order of a hundred million, while the number of jobs is in the order of a million. The training dataset has a few billion examples.

The GDMix model for this problem uses a deep model for fixed effects and logistic regression for random effect. For the deep model, we include a set of textual features based on user profiles and job descriptions. A convolutional neural network similar to the one proposed in [12] is trained via the DeText [11] framework. The random-effect hyper-parameters are kept the same as the baseline

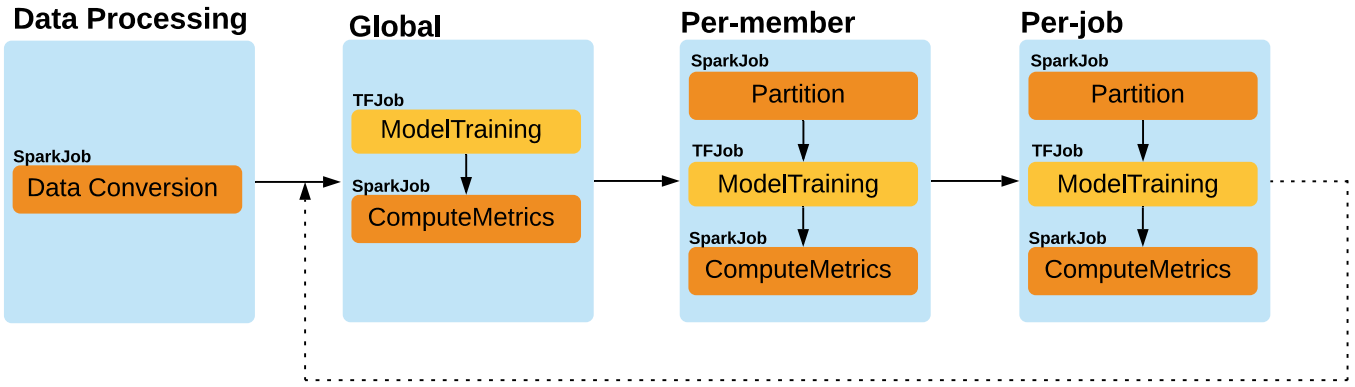



**Figure 2: An example flow consists of one fixed effects model called "global" and two random effects models called "per-member" and "per-job", respectively. The closed loop indicates multiple iterations**

GLMix model. The GDMix-deep model, due to better modeling and representation capacity, exceeds the baseline AUC by 3.85%.

## 4.2 Online Experiments

*4.2.1 Advertising.* We deployed the GDMix deep model mentioned in Section 4.1.1 to the Ads CTR system, and compared it to the existing baseline GLMix production model. Among different business metrics, we focused on CTR and cost-of-click for advertisers (CPC). The A/B test resulted in a +4.4% increase in CTR and -4.2% drop in CPC. Both changes are considered so significant that the GDMix model has now been deployed as the main model in production, replacing the older GLMix model.

*4.2.2 Feed.* The GDMix model described in Section 4.1.2 was tested against the baseline GLMix model in an A/B test. Since both models are identical in structure, relevance metrics were expectedly similar for both models. However, the GDMix model significantly outperformed the GLMix model for freshness - a +0.6% improvement in feed updates viewed for items aged 0-6 hours and a +1.9% improvement in updates viewed for items aged 12-24 hours. At the same time, there was also a +1.87% boost in feed posts getting one or more likes within 24 hours of creation. Thus, GDMix is now able to surface more recent posts to users, resulting in improved user experience.

# 5 EXPERIMENTS ON PUBLIC DATASETS

In this section we report experiments on two public datasets to demonstrate the capabilities of the GDMix training framework.

## 5.1 MovieLens-100K

*5.1.1 Dataset Description.* Movielens-100K dataset is a widely used dataset for recommender systems [9]. The datasets consists of 100K ratings (1-5) by users (943 in number) for movies (1682 in number). Each user has rated at least 20 movies. Features include demographic information for the users (age, gender, occupation, zip). The dataset is rich enough to allow comparison between different algorithms, and small enough for quick experimentation. More details about this dataset can be found at the grouplens website.

*5.1.2 Experiment Setup.* We first convert the rating score to a binary label by converting scores 3 and above to 1s (positive rating), and the rest to 0s (negative rating). There are two groups of features: user features include age, gender and occupation and movie features include genre and release date. For the baseline, we test the GLMix model (logistic regression for both fixed effects and random effects models). The fixed effects model uses both user and movie features, followed by per-user and per-movie random effects. The per-user random effects model (indexed by user ID) uses movie features and the per-movie random effects model (indexed by movie ID) leverages user features.

To test GDMix, we replace the logistic regression fixed effect with a combination of a linear model and a DNN. We use the DeText framework [11] for this purpose. The movie title is added as a text-based feature. The words are converted to embeddings with a dimensionality of 64, and are then fed to a single layer convolutional neural network (50 filters of size 64x3); followed by a fully-connected layer. The logits from the deep network are added to logits from the existing fixed effects logistic regression model (making the combined model very similar to a wide-and-deep model). The per-user and per-movie models remain the same.

| Effects | GLMix | GDMix |
|---|---|---|
| Fixed effects (FE) | 0.6237 | 0.7090 |
| FE + Per-user RE | 0.7058 | 0.7665 |
| FE + Per-user RE + Per-movie RE | 0.7599 | 0.7680 |

**Table 1: Test AUC for GLMix and GDMix models. RE = random effects. Different rows indicate AUC numbers for various combinations of fixed and random effects. For GLMix, all effects are logistic regression model, while for GDMix, the fixed effects use a wide-and-deep model.**

*5.1.3 Experiment Results.* Table 1 contains results for both GLMix and GDMix-based models. Even though the GLMix fixed effect model contains all the features, the random effect model provided extra metric lift due to personalization. The per-user and per-movie models each contribute to large metric improvement. Furthermore, adding a simple CNN model to the existing logistic regression can

boost the fixed effect performance significantly. In this case, adding per-user models still increases the AUC but per-movie models bring diminishing returns. We also observe similar behavior in our production model, i.e. a strong fixed effects model usually leads to lower gains from the random effect models, although the overall AUC still improves.

## 5.2 Criteo Dataset

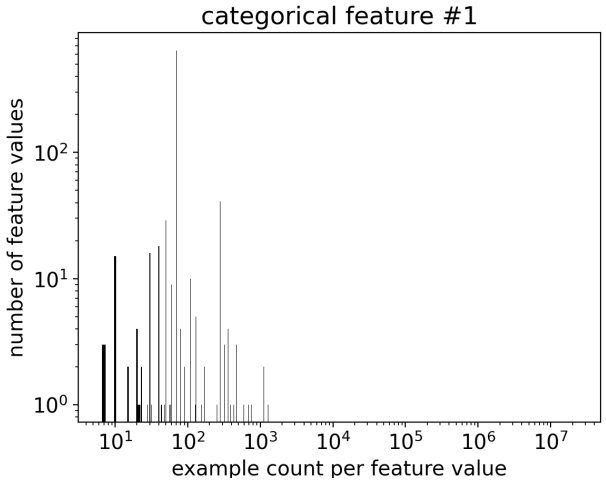
*5.2.1 Dataset Description.* The Kaggle Display Advertising Challenge Dataset [5] consists of a portion of Criteo's advertising traffic over a period of 7 days. Each row corresponds to a display ad served by Criteo. There are about 46 million chronologically ordered examples, consisting of 13 numerical features (integer valued) and 26 categorical features (string valued).

*5.2.2 Experiment Setup.* For the fixed effects model, we leverage the DLRM model [14] and use the network configuration and default parameters used in the code open sourced by the authors. The bottom MLP has 5 hidden layers with width 13, 512, 256, 64 and 16. The top MLP has 3 hidden layers with width 512, 256 and 1. The categorical feature values are mapped to embeddings of size 16.

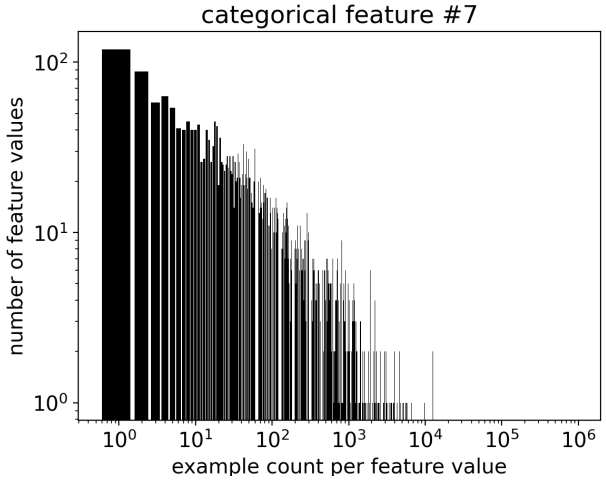
We use logistic regression for the random effects models. Since the Kaggle dataset is anonymized, it is not easy to decide which feature is an entity ID (e.g. user ID or advertiser ID, etc.) on which we can build the random effects. We examine the histogram of example counts for each feature value for all the feature columns. The histograms for categorical feature #1 and #7 (corresponding to column #14 and #20 in the original dataset) are shown in Figure 3. For a feature to be useful for random effects, each of its values should have enough examples, preferably larger than 20. The number of values that have a handful of examples should be as small as possible since these values won't have a reliable random effects model. Among all the features, we find categorical feature #7 to be a good candidate for random effects. Thus, we name the model as the "per-feature-7" model. It has been shown empirically that using all the existing numerical and categorical features in the random effects logistic regression models is not useful to boost the overall AUC. We use a single intercept instead. The random effects models basically account for the popularity of each feature value.

Our GDMix model adds random effects training to the DLRM model. A direct comparison between GDMix and DLRM is not fair since GDMix introduces extra computation and thus is expected to perform better. For Ads use cases, model freshness is very important. A frequent update of the model usually brings metrics improvement. So we compare different models in the re-training scenario. In this experiment, we compare three cases: (1) static DLRM model, the DLRM model is trained once, then tested on the subsequent days. (2) re-training DLRM model, the DLRM model is trained daily and tested on the next day. (3) partial re-training GDMix model, the fixed effect DLRM model is trained once and kept static, then the random effects models are trained once per day then tested on the next day. Our goal is to show that with frequent training of simpler random effect models, we can maintain the AUC without incurring the heavy computational burden of re-training the entire DLRM model.

*5.2.3 Experiment Results.* Table 2 shows the results for the previously mentioned training schemes. We observe nearly constant AUC numbers through the days even for the static DLRM scheme, which is different from observations in our own Ads production pipeline [16]. This suggests that either the dataset was collected for a period of relatively stable traffic or it was sampled in a way to reduce the occurrence of new users or items. Nevertheless, both re-training schemes exhibit slight improvement over the static scheme, except for Day 3. The GDMix scheme only re-trains the intercept-only logistic regression model which can be done within 5 minutes with 50 CPU workers due to high parallelism. On the other hand, full training of the DLRM model takes 2 hours on a single GPU. Even with more GPUs, full training cannot match the speed of training of the logistic regression models. Faster training speed on inexpensive CPU nodes is important in production as CPU compute resources are usually not a limiting factor.




**(a) Histogram of example counts for categorical feature #1**




**(b) Histogram of example counts for categorical feature #7**

**Figure 3: Feature values and their example counts**

| Training Scheme | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 |
|---|---|---|---|---|---|
| Static DLRM | 0.7943 | 0.7942 | 0.7986 | 0.7954 | 0.7926 |
| re-training DLRM | 0.7942 | 0.7929 | 0.7996 | 0.7959 | 0.7938 |
| Partial re-training GDMix | 0.7950 | 0.7943 | 0.7992 | 0.7959 | 0.7933 |

**Table 2: Test AUC for three training schemes. For the static DLRM, the model is trained on Day 0 and Day 1, then tested on Day 2 to 6. For the re-trained DLRM, the model is trained on a sliding window of 2 days and tested on the next day, i.e. trained on Day 0 and 1 tested on Day 2, then trained on Day 1 and 2 and tested on Day 3, etc. For the partially re-trained GDMix, the fixed effects DLRM is trained on Day 0 and Day 1, then kept fixed. The random effects are trained on Day 0 and Day 1 tested on Day 2, then trained on Day 2 tested on Day 3, etc. The fixed effects logits are used for random effects model training and inference.**

## 6 CONCLUSION

Mixed effects models are a viable solution for the personalized recommendation problem. In this paper, we propose deep learning-based mixed effect models and present a training framework for such models. Extensive offline experiments on public and internal datasets show the models achieve better relevance metrics than linear mixed effect models, especially on large scale datasets. Online experiments for LinkedIn Ads and Feed show that, in addition to bringing in relevance metric gains, GDMix is especially suited for model-freshness sensitive applications.

## 7 ACKNOWLEDGMENTS

We would like to thank Jun Jia, Bo Long, Liang Zhang, Bee-Chung Chen, Keerthi Selvaraj and Deepak Agarwal for their guidance at various stages of the development of GDMix. We also appreciate Sida Wang from the AI foundation team, Ruoyan Wang from the Ads AI team, Wei Lu from the Feed AI team and Joojay Huyn from the Learning AI team for their close collaboration during the GDMix onboarding process.

## REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Badi H Baltagi. 2008. *Econometric Analysis of Panel Data.* Wiley.

[3] R. H. Byrd, P. Lu, and J. Nocedal. 1995. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Statist. Comput.* 16, 5 (1995), 1190–1208.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrad, Wei Chai, Mustafa Ispir, and Rohan Anil. 2016. Wide Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).* Association for Computing Machinery, New York, NY, United States, Article 7, 4 pages.

[5] Criteo. 2014. Kaggle Display Advertising Challenge Dataset. https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

[6] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[7] Andrew Gelman. 2005. Analysis of Variance-Why it is more important than ever. In *The Annals of Statistics*, Vol. 33. Institute of Mathematical Statistics, 1–53.

[8] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *CoRR* abs/1909.11810 (2019). https://arxiv.org/abs/1909.11810

[9] Grouplens. 1998. MovieLens 100K Dataset. https://grouplens.org/datasets/movielens/100k/

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence.* 7 pages.

[11] Weiwei Guo, Xiaowei Liu, Sida Wang, Huiji Gao, and Bo Long. 2020. *DeText: A Deep NLP Framework for Intelligent Text Understanding.* https://engineering.linkedin.com/blog/2020/open-sourcing-detext

[12] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019). https://arxiv.org/abs/1906.00091

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[16] Rohan Ramanath, Konstantin Salomatin, Jeffrey D. Gee, Kirill Talanine, Onkar Dalal, Gungor Polatkan, Sara Smoot, and Deepak Kumar. 2021. Lambda Learner: Fast Incremental Learning on Data Streams. In *KDD 2021.* 3492–3502.

[17] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining.* IEEE, 995–1000.

[18] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[19] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2019. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. *CoRR* abs/1909.02107 (2019). https://arxiv.org/abs/1909.02107

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[21] XianXing Zhang, Yitong Zhou, Yiming Ma, Bee-Chung Chen, Liang Zhang, and Deepak Agarwal. 2016. Generalized linear mixed models for large-scale response prediction. In *KDD 2016.*

[22] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence.* Association for the Advancement of Artificial Intelligence, 5941–5948.

[23] Guorui Zhou, Xiaoqiang Zhu, Chengru 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 *KDD 2018.* 1059–1068.