LiRank: Industrial Large Scale Ranking Models at LinkedIn

Fedor Borisyuk, Mingzhou Zhou, Qingquan Song,Siyu Zhu, Birjodh Tiwana,
Ganesh Parameswaran, Siddharth Dangi, Lars Hertel, Qiang Xiao, Xiaochen Hou, Yunbo Ouyang,
Aman Gupta, Sheallika Singh, Dan Liu, Hailing Cheng, Lei Le, Jonathan Hung, Sathiya Keerthi,
Ruoyan Wang, Fengyu Zhang, Mohit Kothari, Chen Zhu, Daqi Sun, Yun Dai, Xun Luan, Sirou Zhu,
Zhiwei Wang, Neil Daftary, Qianqi Shen, Chengming Jiang, Haichao Wei, Maneesh Varshney,
Amol Ghoting, Souvik Ghosh
LinkedIn Inc.

ABSTRACT

We present *LiRank*, a large-scale ranking framework at LinkedIn that brings to production state-of-the-art modeling architectures and optimization methods. We unveil several modeling improvements, including Residual DCN, which adds attention and residual connections to the famous DCNv2 architecture. We share insights into combining and tuning SOTA architectures to create a unified model, including Dense Gating, Transformers and Residual DCN. We also propose novel techniques for calibration and describe how we productionalized deep learning based explore/exploit methods.

To enable effective, production-grade serving of large ranking models, we detail how to train and compress models using quantization and vocabulary compression. We provide details about the deployment setup for large-scale use cases of Feed ranking, Jobs Recommendations, and Ads click-through rate (CTR) prediction.

We summarize our learnings from various A/B tests by elucidating the most effective technical approaches. These ideas have contributed to relative metrics improvements across the board at LinkedIn: +0.5% member sessions in the Feed, +1.76% qualified job applications for Jobs search and recommendations, and +4.3% for Ads CTR. We hope this work can provide practical insights and solutions for practitioners interested in leveraging large-scale deep ranking systems.

CCS CONCEPTS

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

KEYWORDS

Large Scale Ranking, Deep Neural Networks

ACM Reference Format:
Fedor Borisyuk, Mingzhou Zhou, Qingquan Song,Siyu Zhu, Birjodh Tiwana,, Ganesh Parameswaran, Siddharth Dangi, Lars Hertel, Qiang Xiao, Xiaochen Hou, Yunbo Ouyang, Aman Gupta, Sheallika Singh, Dan Liu, Hailing Cheng, Lei Le, Jonathan Hung, Sathiya Keerthi,, Ruoyan Wang, Fengyu Zhang, Mohit Kothari, Chen Zhu, Daqi Sun, Yun Dai, Xun Luan, Sirou Zhu, Zhiwei Wang, Neil Daftary, Qianqi Shen, Chengming Jiang, Haichao Wei, Maneesh Varshney,, Amol Ghoting, Souvik Ghosh. 2018. LiRank: Industrial Large Scale Ranking Models at LinkedIn. In *Proceedings of (KDD'24)*. ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

LinkedIn is the world's largest professionals network with more than 1 billion members in more than 200 countries and territories worldwide. Hundreds of millions of LinkedIn members engage on a regular basis to find opportunities and connect with other professionals.

At LinkedIn, we strive to provide our members with valuable content that can help them build professional networks, learn new skills, and discover exciting job opportunities. To ensure this content is engaging and relevant, we aim to understand each member's specific preferences. This may include interests such as keeping up with the latest news and industry trends, participating in discussions by commenting or reacting, contributing to collaborative articles, sharing career updates, learning about new business opportunities, or applying for jobs.

In this paper, we introduce a set of innovative enhancements to model architectures and optimization strategies, all aimed at enhancing the member experience. The **contribution of the paper** consists of:

- We propose a novel Residual DCN layer (§3.3), an improvement on top of DCNv2[33], with attention and residual connections.
- We propose a novel isotonic calibration layer trained jointly within deep learning model (§3.4).
- We provide customizations of deep-learning based exploit/explore methods to production (§3.8).
- Integrating various architectures into a large-scale unified ranking model presented challenges such as diminishing returns (first attempt lead to no gain), overfitting, divergence, and different gains across applications. In §3, we discuss our approach to developing high-performing production ranking models, combining Residual DCN (§3.3), isotonic calibration layer (§3.4), dense gating with larger MLP (§3.5), incremental training (§3.6), transformer-based history modeling (§3.7), deep learning explore-exploit strategies (§3.8), wide popularity features (§3.9), multi-task learning (§3.10), dwell modeling (§3.11).
- We share practical methods to speed up training process, enabling rapid model iteration (§4).
- We provide insights into training and compressing deep ranking models using quantization (§3.13) and vocabulary compression

KDD'24, August 25–29, 2024, Barcelona, Spain

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

(§3.12) to facilitate the effective deployment of large-ranking models in production.

Proposed modeling advancements within this paper enabled our models to efficiently handle a larger number of parameters, leading to higher-quality content delivery. Within the paper we introduce details of large scale architectures of Feed ranking in §3.1, Ads CTR model §3.2, and Job recommendation ranking models in §5.3.

In §5, we detail our experiences in deploying large-ranking models in production for Feed Ranking, Jobs Recommendations, and Ads CTR prediction, summarizing key learnings gathered from offline experimentation and A/B tests. Notably, the techniques presented in this work have resulted in significant relative improvements: a 0.5% increase in Feed sessions, a 1.76% enhancement in the number of qualified applicants within Job Recommendations, and a 4.3% boost in Ads CTR. We believe that this work can provide practical solutions and insights for engineers who are interested in applying large DNN ranking models at scale.

2 RELATED WORK

The use of deep neural network models in personalized recommender systems has been dominant in academia and industry since the success of the Wide&Deep model[5] in 2016. Typically, these models consist of feature embeddings, feature selection, and feature interaction components, with much research focused on enhancing feature interactions. Tge Wide&Deep model[5] initiated this trend by combining a generalized linear model with an MLP network. Subsequent research aimed to keep the MLP network for implicit feature interactions and replace the linear model with other modules for capturing explicit higher-order feature interactions. Examples include DeepFM[12], which replaced the linear model with FM; deep cross network (DCN)[33] and its follow-up DCNv2[35], which introduced a cross network for high-order feature interactions; xDeepFM[18], offering compressed interaction network (CIN) for explicit vector-wise feature interactions; AutoInt[29], which introduced self-attention networks for explicit feature interaction; AFN[6], exploring adaptive-order feature interactions through a logarithmic transformation layer; and FinalMLP[20], which achieved impressive performance by combining two MLPs.

We experimented with and customized these architectures for various LinkedIn recommender tasks, with DCNv2 proving to be the most versatile. We propose enhancements to DCNv2, referred to as Residual DCN, in this paper. Additionally, we implemented a model parallelism design in TensorFlow(TF), similar to the approach proposed in the DLRM[21] paper, to accelerate model training with large embedding tables.

In our investigation, we've encountered challenges when attempting to seamlessly integrate original architectures into production environments. These challenges often manifest as issues such as model training divergence, over-fitting, or limited observable performance improvements. Crafting a high-performing model by effectively leveraging these architectures demands substantial effort, often characterized by a painstaking process of trial and error. Consequently, in this paper, we aim to offer valuable insights derived from our experiences in successfully assembling state-of-the-art (SOTA) architectures into production-ready ranking models.

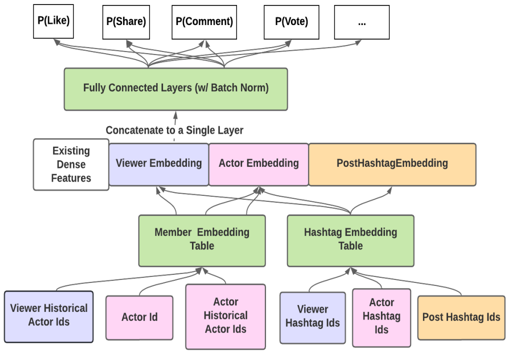

Figure 1: Contribution tower of the main Feed ranking model

While enhancing neural network predictive performance through various optimizations and architectures, the space of calibration remained relatively stable. Traditional industry-standard methods [10] like Histogram binning, Platt Scaling, and Isotonic Regression are applied in post-processing steps after deep model training. Some research has introduced calibration-aware losses to address under/over calibration issues usually resulting in trade-off [11] or slight improved metrics [2]. In §3.4 we propose an isotonic calibration layer within the deep learning model which learns to calibrate deep model scores during model training and improves model predictive accuracy significantly.

3 LARGE RANKING MODELS

In this section, we introduce large ranking models used by LinkedIn Feed Ranking and Ads CTR (click-through-rate) prediction. We observe that the choice of architecture components varies based on the use case. We'll share our insights on building effective ranking models for production scenarios.

3.1 Feed Ranking Model

The primary Feed ranking model employs a point-wise ranking approach, predicting multiple action probabilities including like, comment, share, vote, and long dwell and click for each <member, candidate post> pair. These predictions are linearly combined to generate the final post score. A TF model with a multi-task learning (MTL) architecture generates these probabilities in two towers: the click tower for probabilities of click and long dwell, and contribution tower for contribution and related predictions. Both towers use the same set of dense features normalized based on their distribution[13], and apply multiple fully-connected layers. Sparse ID embedding features (§A.1) are transformed into dense embeddings [22] through lookup in embedding tables of Member/Actor and Hashtag Embedding Table as in Figure 1. For reproducability in appendix in Figure 8 we provide a diagram showing how different architectures are connected together into a single model.

3.2 Ads CTR Model

At LinkedIn, ads selection relies on click-through-rate (CTR) prediction, estimating the likelihood of member clicks on recommended ads. This CTR probability informs ad auctions for displaying ads to

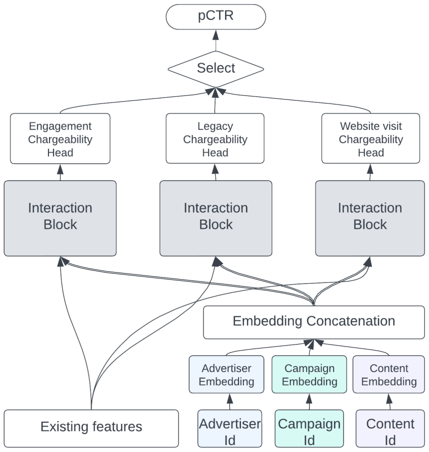

Figure 2: Ads CTR chargeability-based multi-task model

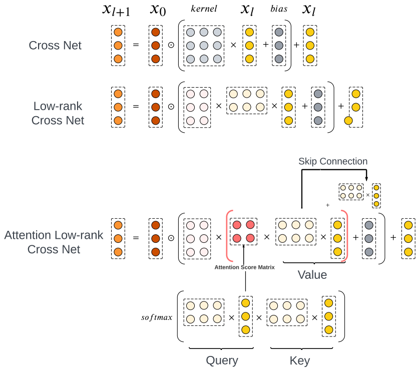

Figure 3: Residual Cross Network

members. Advertisers customize chargeable clicks for campaigns, such as some advertisers consider social interaction such as 'like', 'comment' as chargeable clicks while others only consider visiting ads websites as clicks. Usually only positive customized chargeable clicks are treated as positive labels. To better capture user interest, our CTR prediction model is a chargeability-based MTL model with 3 heads that correspond to 3 chargeability categorizations where similar chargeable definitions are grouped together regardless of advertiser customization. Each head employs independent interaction blocks such as MLP and DCNv2 blocks. The loss function combines head-specific losses. For features, besides traditional features from members and advertisers, we incorporate ID features to represent advertisers, campaigns, and advertisements. The model architecture is depicted in Figure 2.

3.3 Residual DCN

To automatically capture feature interactions, we utilized DCNv2 [34]. Our offline experiments revealed that two DCNv2 layers provided sufficient interaction complexity, as adding more layers yielded diminishing relevance gains while increasing training and serving times significantly. Despite using just two layers, DCNv2 added a considerable number of parameters due to the large feature input dimension. To address this, we adopted two strategies for enhancing efficiency. First, following [34], we replaced the weight matrix with two skinny matrices resembling a low-rank approximation. Second, we reduced the input feature dimension by replacing sparse one-hot features with embedding-table look-ups, resulting in nearly a 30% reduction. These modifications allowed us to substantially reduce DCNv2's parameter count with only minor effects on relevance gains, making it feasible to deploy the model on CPUs.

To further enhance the power of DCNv2, specifically, the cross-network, introduced an attention schema in the low-rank cross net. Specifically, the original low-rank mapping is duplicated as three with different mapping kernels, where the original one serves as the value matrix and the other two as the query and key matrices, respectively. An attention score matrix is computed and inserted between the low-rank mappings. Figure 3 describes the basic scaled dot-product self-attention. A temperature could also be added to balance the complicacy of the learned feature interactions. In the extreme case, the attention cross net will be degenerated to the normal cross net when the attention score matrix is an identity matrix. Practically, we find that adding a skip connection and fine-tuning the attention temperature is beneficial for helping learn more complicated feature correlations while maintain stable training. By paralleling a low-rank cross net with an attention low-rank cross net, we found a statistically significant improvement on feed ranking task (§5.2).

3.4 Isotonic Calibration Layer in DNN

Model calibration ensures that estimated class probabilities align with real-world occurrences, a crucial aspect for business success. For example, Ads charging prices are linked to click-through rate (CTR) probabilities, making accurate calibration essential. It also enables fair comparisons between different models, as the model score distribution can change when using different models or objectives. Traditionally, calibration is performed post-training using classic methods like Platt scaling and isotonic regression. However, these methods are not well-suited for deep neural network models due to limitations like parameter space constraints and incompatibility. Additionally, scalability becomes challenging when incorporating multiple features like device, channel, or item IDs into calibration.

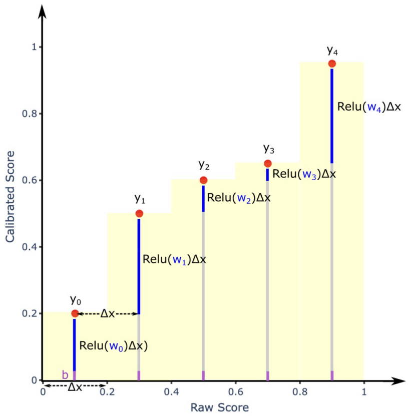

Figure 4: Isotonic layer representation

To address the issues mentioned above, we developed a customized isotonic regression layer (referred as *isotonic layer*) that can be used as a native neural network layer to be co-trained with a deep neural network model to perform calibration. Similar to the isotonic regression, the isotonic layer follows the piece-wise fitting idea. It bucketizes the predicted values (probabilities must be converted back to logits) by a given interval v_i and assigns a trainable weight w_i for each bucket, which are updated during the training with other network parameters (Figure 4). The isotonic property is guaranteed by using non-negative weights, which is achieved by using the Relu activation function. To enhance its calibration power with multiple features, the weights can be combined with an embedding representation (a vector whose element is denoted as e_i) that derives from all calibration features. Finally we obtain

$$y_{cali} = \Sigma_{i=0}^{i=k} Relu(e_i + w_i) \cdot v_i + b, v_i = \begin{cases} step, & \text{if } i < k \\ y - step \cdot k, & \text{i=k} \end{cases}, \\ k = \arg\max_j (y - step \cdot j > 0). \tag{1}$$

3.5 Dense Gating and Large MLP

Introducing personalized embeddings to global models helps introduce interactions among existing dense features, most of them being multi-dimensional count-based and categorical features. We flattened these multi-dimensional features into a singular dense vector, concatenating it with embeddings before transmitting it to the MLP layers for implicit interactions. A straightforward method to enhance gain was discovered by enlarging the width of each MLP layer, fostering more comprehensive interactions. For Feed, the largest MLP configuration experimented with offline was 4 layers of width 3500 each (refer as "Large MLP", or LMLP). Notably, gains manifest online exclusively when personalized embeddings are in play. However, this enhancement comes at the expense of increased scoring latency due to additional matrix computations. To address this issue, we identified a optimal configuration that maximizes gains within the latency budget.

Later, inspired by Gate Net [14], we introduced a gating mechanism to hidden layers. This mechanism regulates the flow of information to the next stage within the neural network, enhancing the learning process. We found that the approach was most cost-effective when applied to hidden layers, introducing only negligible extra matrix computation while consistently producing online lift.

Additionally we have explored sparse gated mixture of expert models (sMoE) [26]. We report ablation studies in §5.2.

3.6 Incremental Training

Large-scale recommender systems must adapt to rapidly evolving ecosystems, constantly incorporating new content such as Ads, news feed updates, and job postings. To keep pace with these changes, there is a temptation to use the last trained model as a starting point and continue training it with the latest data, a technique known as *warm start.* While this can improve training efficiency, it can also lead to a model that forgets previously learned information, a problem known as catastrophic forgetting[8]. Incremental training, on the other hand, not only uses the previous model for weight initialization but also leverages it to create an informative regularization term.

Denote the current dataset at timestamp t as $\mathcal{D}_t$, the last estimated weight vector as $\mathbf{w}_{t-1}$, the Hessian matrix with regard to $\mathbf{w}_{t-1}$ as $\mathcal{H}_{t-1}$. The total loss up to timestamp t is approximated as

$$\text{loss}_{\mathcal{D}_t}(\mathbf{w}) + \lambda_f/2 \times (\mathbf{w} - \mathbf{w}_{t-1})^T \mathcal{H}_{t-1}(\mathbf{w} - \mathbf{w}_{t-1}), \tag{2}$$

where λ_f is the forgetting factor for adjusting the contribution from the past samples. In practice $\mathcal{H}_{t-1}$ will be a very large matrix. Instead of computing $\mathcal{H}_{t-1}$, we only use the diagonal elements $\text{diag}(\mathcal{H}_{t-1})$, which significantly reduces the storage and the computational cost. For large deep recommendation models, since the second order derivative computation is expensive, Empirical Fisher Information Matrix (FIM) [16, 24] is proposed to approximate the diagonal of the Hessian.

A typical incremental learning cycle consists of training one initial cold start model and training subsequent incrementally learnt models. To further mitigate catastrophic forgetting and address this issue, we use both the prior model and the initial cold start model to initialize the weights and to calculate the regularization term. In this setting, the total loss presented in (2) is:

$$\begin{aligned} &\text{loss}_{\mathcal{D}_t}(\mathbf{w}) + \lambda_f/2 \times [\alpha(\mathbf{w} - \mathbf{w}_0)^T \mathcal{H}_0(\mathbf{w} - \mathbf{w}_0) \\ &+ (1 - \alpha)(\mathbf{w} - \mathbf{w}_{t-1})^T \mathcal{H}_{t-1}(\mathbf{w} - \mathbf{w}_{t-1})], \end{aligned} \tag{3}$$

where $\mathbf{w}_0$ is the weight of the initial cold start model and $\mathcal{H}_0$ is the Hessian with regard to $\mathbf{w}_0$ over the cold start training data. Model weight $\mathbf{w}$ is initialized as $\alpha\mathbf{w_0} + (1-\alpha)\mathbf{w}_{t-1}$. The additional tunable parameter $\alpha \in [0, 1]$ is referred to as *cold weight* in this paper. Positive cold weight continuously introduces the information of the cold start model to incremental learning. When cold weight is 0, then equation (3) is the same as (2).

3.7 Member History Modeling

To model member interactions with platform content, we adopt an approach similar to [4, 36]. We create historical interaction sequences for each member, with item embeddings learned during optimization or via a separate model, like [23]. These item embeddings are concatenated with action embeddings and the embedding of the item currently being scored (early fusion). A two-layer Transformer-Encoder [31] processes this sequence, and the max-pooling token is used as a feature in the ranking model. To enhance information, we also consider the last five sequence steps, flatten and concatenate them as additional input features for the ranking model. To reduce latency, we experimented with shorter sequences and smaller feed-forward network dimensions within the Transformer. In ablation experiments in §5.2 we refer to history modeling as TransAct.

Our findings show that a two-layer transformer with a feed-forward dimension equal to half the input embedding size delivers most relevance gains. While longer sequences improve relevance metrics, the added training and serving time did not justify extended history sequences.

3.8 Explore and Exploit

The exploration vs exploitation dilemma is common in recommender systems. A simple utilization of member's historical feedback data ("exploitation") to maximize immediate performance might hurt long term gain; while boosting new items ("exploration") could help improve future performance at the cost of short term gain. To balance them, the traditional methods such as Upper Confidence Bounds (UCB) and Thompson sampling are utilized, however, they can't be efficiently applied to deep neural network models. To reduce the posterior probability computation cost and maintain certain representational power, we adopted a method similar to the Neural Linear method mentioned in the paper [25], namely we performed a Bayesian linear regression on the weights of the last layer of a neural network. Given a predicted value y_i for each input x_i is given by $y_i = WZx$, where W is the weights of last layer and Zx is the input to the last layer given input x. Given W we apply a Bayesian linear regression to y with respect to Zx, and acquire the posterior probability of W, which is fed into Thompson Sampling. Unlike the method mentioned in the paper, we don't independently train a model to learn a representation for the last layer. The posterior probability of W is incrementally updated at the end of each offline training in a given period, thus frequent retrainings would capture new information timely. The technique has been applied to feed and online A/B testing showed relative +0.06% professionals Daily Active Users.

3.9 Wide Popularity Features

Our ranking model combines a global model with billions of parameters to capture broad trends and a random effect model to handle variations among individual items, assigning unique values reflecting their popularity among users. Due to our platform's dynamic nature, random effect models receive more frequent training to adapt to shifting trends.

For identifiers with high volatility and short-lived posts, known as Root Object ID, we use a specialized Root-object (RO) model. This model is trained every 8 hours with the latest data to approximate the residuals between the main model's predictions and actual labels. Due to higher coverage of labels we used Likes and Clicks within RO model.

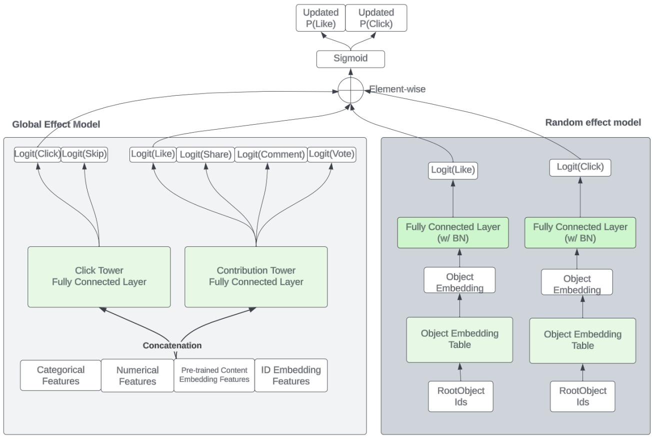

Figure 5: RO Wide model on click and like towers.

The final prediction of our model, denoted as y_{final}, hinges on the summation of logits derived from the global model and the random effect model. It is computed as follows:

$$y_{\text{final}} = \sigma\left(\text{logit}(y_{\text{global\_effect}}) + \text{logit}(y_{\text{random\_effect}})\right),$$

where σ signifies the sigmoid function.

Large embedding tables aid our item ID learning process. We've incorporated an explore/exploit algorithm alongside RO Wide scores, improving the Feed user experience with +0.17% relative increase in engaged DAU (daily active users).

3.10 Multi-task Learning

Multi-task Learning (MTL) is pivotal for enhancing modern feed ranking systems, particularly in Second Pass Ranking (SPR). MTL enables SPR systems to optimize various ranking criteria simultaneously, including user engagement metrics, content relevance, and personalization. Our exploration of MTL in SPR has involved various model architectures designed to improve task-specific learning, each with unique features and benefits: (1) Hard Parameter Sharing: involves sharing parameters directly across tasks, serving as a baseline, (2) Grouping Strategy: tasks are grouped based on similarity, such as positive/negative ratio or semantic content. For example, tasks like 'Like' and 'Contribution' are can be grouped together into a single tower supporting both tasks due to their higher positive rates, while 'Comment' and 'Share' are grouped separately with lower positive rates. We also explored common approaches, including MMoE [19] and PLE [30]. In our experiments, the Grouping Strategy showed a modest improvement in metrics with only a slight increase in model parameters (see Table 1). On the other hand, MMoE and PLE, while offering significant performance boosts, expanded the parameter count by 3x-10x, depending on the expert configuration, posing challenges for large-scale online deployment.

Model	Contributions
Hard Parameter Sharing	baseline
Grouping Strategy	+0.75%
MMoE	+1.19%
PLE	+1.34%

Table 1: Performance comparison of MTL models

3.11 Dwell Time Modeling

Dwell time, reflecting member content interaction duration, provides valuable insights into member's behavior and preferences. We introduced a 'long dwell' signal to detect passive content consumption on the LinkedIn Feed. Implementing this signal effectively, allows the capture of passive but positive engagement. Modeling dwell time presented technical challenges: (1) Noisy dwell time data made direct prediction or logarithmic prediction unsuitable due to high volatility, (2) Static threshold identification for 'long dwell' couldn't adapt to evolving user preferences, manual thresholds lacked consistency and flexibility, (3) Fixed thresholds could bias towards content with longer dwell times, conflicting with our goal of promoting engaging posts across all content types on LinkedIn Feed.

To address these challenges, we designed a 'long dwell' binary classifier predicting whether there is more time spent on a post than a specific percentile (e.g., 90th percentile). Specific percentiles are determined based on contextual features such as ranking position, content type, and platform, forming clusters for long-dwell threshold setting and enhancing training data. By daily measuring cluster distributions, we capture evolving member consumption patterns and reduce bias and noise in the dwell time signal. The model operates within a Multi-task multi-class framework, resulting in relative improvements of a 0.8% in overall time spent, a 1% boost in time spent per post, and a 0.2% increase in member sessions.

3.12 Model Dictionary Compression

The traditional approach to mapping high-dimensional sparse categorical features to an embedding space involves two steps. First, it converts string-based ID features to integers using a static hashtable. Next, it utilizes a memory-efficient Minimal Perfect Hashing Function (MPHF) [3] to reduce in-memory size. These integer IDs serve as indices for accessing rows in the embedding matrix, with cardinality matching that of the static hashtable or unique IDs in the training data, capped at a maximum limit. The static hashtable contributes for about 30% of memory usage, which can become inefficient as vocabulary space grow and the vocabulary-to-model size ratio increases. Continuous training further complicates matters, as it demands incremental vocabulary updates to accommodate new data.

QR hashing [28] offers a solution by decomposing large matrices into smaller ones using quotient and remainder techniques while preserving embedding uniqueness across IDs. For instance, a vocabulary of 4 billion with a 1000x compression ratio in a QR strategy results in two tiny embedding matrices of approximately 4 million rows in sum — roughly 4 million from the quotient matrix and around 1000 from the remainder matrix. This approach has demonstrated comparable performance in offline and online metrics in Feed/Ads. We found that sum aggregation worked the best, while multiplication aggregation suffered from convergence issues due to numerical precision, when embeddings are initialized close to 0. QR hashing's compatibility with extensive vocabulary opens doors to employing a collision-resistant hashing function like MurmurHash, potentially eliminating vocabulary maintenance. It also generates embedding vectors for every training item ID, resolving the Out-of-Vocabulary (OOV) problem and can potentially capture more diverse signals from the data. Refer Figure 9 in Appendix for illustration on the technique.

3.13 Embedding Table Quantization

Embedding tables, often exceeding 90% of a large-scale deep ranking model's size, pose challenges with increasing feature, entity, and embedding dimension sizes. These components can reach trillions of parameters, causing storage and inference bottlenecks due to high memory usage [9] and intensive lookup operations. To tackle this, we explore embedding table quantization, a model dictionary compression method that reduces embedding precision and overall model size. For example, using an embedding table of 10 million rows by 128 with fp32 elements, 8-bit row-wise min-max quantization [27] can reduce the table size by over 70%. Research has shown that 8-bit post-training quantization maintains performance and inference speed without extra training costs or calibration data requirements [9], unlike training-aware quantization. To ensure quick model delivery, engineer flexibility, and smooth model development and deployment, we opt for post-training quantization, specifically employing middle-max row-wise embedding-table quantization. Unlike min-max row-wise quantization which saves the minimum value and the quantization bin-scale value of each embedding row, middle-max quantization saves the middle values of each row defined by $\mathrm{X}_{i,:}^{middle} = \frac{\mathrm{X}_{i,:}^{max} * 2^{bits-1} + \mathrm{X}_{i,:}^{min} * (2^{bits-1}-1)}{2^{bits}-1}$, where $\mathrm{X}_{i,:}^{min}$ and $\mathrm{X}_{i,:}^{max}$ indicate the minimum and maximum value of the i-th row of an embedding table $\mathbf{X}$. The quantization and dequantization steps are described as: $\mathrm{X}_{i,:}^{int} = round(\frac{\mathrm{X}_{i,:} - \mathrm{X}_{i,:}^{middle}}{\mathrm{X}_{i,:}^{scale}})$ and $\mathrm{X}_{i,:}^{dequant} = \mathrm{X}_{i,:}^{middle} + \mathrm{X}_{i,:}^{int} * \mathrm{X}_{i,:}^{scale}$, where $\mathrm{X}_{i,:}^{scale} = \frac{\mathrm{X}_{i,:}^{max} - \mathrm{X}_{i,:}^{min}}{2^{bits}-1}$.

We choose middle-max quantization for two reasons: (1) Embedding values typically follow a normal distribution, with more values concentrated in the middle of the quantization range. Preserving these middle values reduces quantization errors for high-density values, potentially enhancing generalization performance. (2) The range of $\mathrm{X}_{i,:}^{int}$ values falls within $[-128, 127]$, making integer casting operations from float to int8 reversible and avoiding 2's complement conversion issues, i.e., cast(cast(x, int8), int32) may not be equal to x due to the 2's complement conversion if $x \in [0, 255]$. Experimental results show that 8-bit quantization generally achieves performance parity with full precision, maintaining reasonable serving latency even in CPU serving environments with native TF operations. In Ads CTR prediction, we observed a +0.9% CTR relative improvement in online testing, which we attribute to quantization smoothing decision boundaries, improving generalization on unseen data, and enhancing robustness against outliers and adversaries.

4 TRAINING SCALABILITY

During development of large ranking models we optimized training time via set of techniques including 4D Model Parallelism, Avro Tensor Dataset Loader, offloading last-mile transformation to async stage and prefetching data to GPU with significant improvements to training speed (see Table 2). Below we provide descriptions on why and how we developed it.

4.1 4D Model Parallelism

We utilized Horovod to scale out synchronous training with multiple GPUs. During benchmarking, we have observed performance bottlenecks during gradient synchronization of the large embedding tables. We implemented 4D model parallelism in TensorFlow (TF) to distribute the embedding table into different processes. Each worker process will have one specific part of the embedding table shared among all the workers. We were able to reduce the gradient synchronization time by exchanging input features via all-to-all (to share the features related to the embedding lookup to specific workers), which has a lower communication cost compared to exchanging gradients for large embedding tables. From our benchmarks, model parallelism reduced training time from 70 hours to 20 hours.

4.2 Avro Tensor Dataset Loader

We also implemented and open sourced a TF Avro reader that is up to 160x faster than the existing Avro dataset reader according to our benchmarks. Our major optimizations include removing unnecessary type checks, fusing I/O operations (parsing, batching, shuffling), and thread auto-balancing and tuning. With our dataset loader, we were able to resolve the I/O bottlenecks for training job, which is common for large ranking model training. The e2e training time was reduced by 50% according to our benchmark results (Table 2).

4.3 Offload Last-mile Transformation to Asynchronous Data Pipeline

We observed some last-mile in-model transformation that happens inside the training loop (ex. filling empty rows, conversion to Dense, etc.). Instead of running the transformation + training synchronously in the training loop, we moved the non-training related transformation to a transformation model, and the data transformation is happening in the background I/O threads that is happening asynchronously with the training step. After the training is finished, we stitched the two model together into the final model for serving. The e2e training time was reduced by 20% according to our benchmark results (Table 2).

4.4 Prefetch Dataset to GPU

During the training profiling, we saw CPU -> GPU memory copy happens during the beginning of training step. The memory copy overhead became significant once we increased the batch size to larger values (taking up to 15% of the training time). We utilized customized TF dataset pipeline and Keras Input Layer to prefetch the dataset to GPU in parallel before the next training step begins.

Optimization Applied	**e2e Training Time Reduction**
4D Model Parallelism	71%
Avro Tensor Dataset Loader	50%
Offload last-mile transformation	20%
Prefetch dataset to GPU	15%

Table 2: Training performance relative improvements

5 EXPERIMENTS

We conduct offline ablation experiments and A/B tests across various surfaces, including Feed Ranking, Ads CTR prediction, and Job recommendations. In Feed Ranking, we rely on offline replay metrics, which have shown a correlation with production online A/B test results. Meanwhile, for Ads CTR and Job recommendations, we find that offline AUC measurement aligns well with online experiment outcomes.

5.1 Incremental Learning

We tested incremental training on both Feed ranking models and Ads CTR models. The experiment configuration is set in Table 3. We start with a cold start model, followed by a number of incremental training iterations (6 for Feed ranking models and 4 for Ads CTR models). For each incrementally trained model, we evaluate on a fixed test dataset and average the metrics. The baseline is the evaluation metric on the same fixed test dataset using the cold start model.

Table 4 and 5 summarize the metrics improvements and training time improvements for both Feed ranking models and Ads CTR models, after tuning the cold weight and λ. For both models, incremental training boosted metrics with significant training time reduction. Contributions measurement for Feed is explained in §5.2.

Experiments	Feed Ranking	Ads CTR
Cold Start Data Range	21 days	14 days
Incremental Data Range	1 day	0.5 day
Incremental Iterations	6	4

Table 3: Incremental Experiments Settings

	Contributions	Training Time
Cold Start	-	-
Incremental Training	+1.02%	-96%

Table 4: Feed ranking model results summary

	Test AUC	Training Time
Cold Start	-	-
Incremental Training	+0.18%	-96%

Table 5: Ads CTR model results summary

5.2 Feed Ranking

To assess and compare Feed ranking models offline, we employ a "replay" metric that estimates the model's online contribution rate (e.g., likes, comments, re-posts). For evaluation, we rank a small portion of LinkedIn Feed sessions using a pseudo-random ranking model, which uses the current production model to rank all items but randomizes the order of the top N items uniformly. After training a new experimental model, we rank the same sessions offline with it. When a matched impression appears at the top position ("matched imp @ 1," meaning both models ranked the same item at Feed position 1) and the member served the randomized model makes a contribution to that item, we assign a contribution reward to the experimental model: contribution rate = $\frac{\text{\# of matched imps @ 1 with contribution}}{\text{\# of matched imps @ 1}}$

This methodology allows unbiased offline comparison of experimental models [17]. We use offline replay to assess Feed Ranking models, referred to as 'contribution' throughout the paper (Table 6). The table illustrates the impact of various production modeling techniques on offline replay metrics, including Isotonic calibration layer, low-rank DCNv2, Residual DCN, Dense Gating, Large MLP layer, Sparse Features, MTL enhancements, TransAct, and Sparsely Gated MMoE. These techniques, listed in Table 6, are presented in chronological order of development, highlighting incremental improvements. We've deployed these techniques to production, and through online A/B testing, we observed a 0.5% relative increase in the number of member sessions visiting LinkedIn.

5.3 Jobs Recommendations

In Job Search (JS) and Jobs You Might Be Interested In (JYMBII) ranking models, 40 categorical features are embedded through 5 shared embedding matrices for title, skill, company, industry, and seniority. The model predicts probability of P(job application) and P(job click). We adopted embedding dictionary compression described in §3.12 with 5x reduction of number of model parameters, and the evaluation does not show any performance loss compared to using vanilla id embedding lookup table. We also did not observe improvement by using Dense Gating (§3.5) in JYMBII and

Model	Contributions
Baseline	-
+ 30 dimensional ID embeddings (IDs)	+1.89%
+ Isotonic calibration layer	+1.08%
+ Large MLP (LMLP)	+1.23%
+ Dense Gating (DG)	+1.00%
+ Multi-task (MTL) Grouping	+0.75%
+ Low-rank DCNv2 (LDCNv2)	+1.26%
+ TransAct	+1.66%
+ Residual DCN (RDCN)	+2.15%
+ LDCNv2+LMLP+TransAct	+3.45%
+ RDCN+LMLP+TransAct	+3.62%
+ Sparsly Gated MMoE	+4.14%

Table 6: Ablation study of model architecture components in Feed ranking on the relative off-policy measurement.

JS with extensive tuning of models. These entity id embeddings are shared by Job Search and JYMBII Recommendation, and then a task-specific 2-layer DCN is added on top to explicitly capture the feature interactions. Overall we observe significant offline AUC lift of +1.63% for Job Search and 2.10% for JYMBII. For reproducibility purposes we provide model architecture and ablation study of different components of JYMBII and Job Search model in §A.8.

The ranking models with higher AUC shown above also transferred to significant metrics lift in the online A/B testing, leading to relative 1.76% improvement in Qualified Applications across Job Search and JYMBII. Percent Chargeable Views is the fraction of clicks among all clicks on promoted jobs. Qualified Application is the total count of all qualified job applications.

Online Metrics	Job Search	JYMBII
Percent Chargeable Views	+1.70%	+4.16%
Qualified Application	+0.89%	+0.87%

Table 7: Online experiment relative metrics improvements of JS and JYMBII ranking

5.4 Ads CTR

Our baseline model is a multilayer perceptron model that derived from its predecessor GDMix model [15] with proper hyper-parameter tuning. Features fall into five categories: contextual, advertisement, member, advertiser, ad-member interaction. Baseline model doesn't have Id features. In the Table 5 we show relative improvements of each of the techniques including ID embeddings, Quantization, Low-rank DCNv2, TransAct and Isotonic calibration layer. Techniques mentioned in the table are ordered in timeline of development. We have deployed techniques to production and observed 4.3% CTR relative improvement in online A/B tests.

6 DEPLOYMENT LESSONS

Over the time of development we learnt many deployment lessons. Here we present couple of interesting examples.

6.1 Scaling up Feed Training Data Generation

At the core of the Feed training data generation is a join between post labels and features. The labels dataset consists of impressed posts from all sessions. The features dataset exists on a session level. Here, each row contains session-level features and all served posts with their post-level features. To combine these, we explode the features dataset to be on a post-level and join with the labels dataset. However, as Feed scaled up from using 13% of sessions for training to using 100% of sessions, this join caused long delay. To optimize the pipeline we made two key changes that reduced the runtime by 80% and stabilized the job. Firstly, we recognized that not all served posts are impressed. This means the join with the labels dataset drastically reduces the number of rows. Furthermore, exploding the features dataset repeats session-level features for every post. We therefore changed the pipeline to explode only the post features and keys, join with the labels, and add the session-level features in a second join. Despite this resulting in two joins, each join was now smaller and resulted in an overall shuffle write size reduction of 60%. Secondly, we tuned the Spark compression, which resulted in an additional 25% shuffle write size reduction. These changes allowed us to move forward with 100% of sessions for training.

Model	AUC
Baseline	-
ID embeddings (IDs)	+1.27%
IDs+Quantization 8-bit	+1.28%
IDs+DCNv2	+1.45%
IDs+low-rank DCNv2	+1.37%
IDs+isotonic layer	+1.39% (O/E ratio +1.84%)
IDs+low-rank DCNv2+isotonic layer	+1.47%
IDs + TransAct	+2.20%

Table 8: Ablation study of different Ads CTR model architecture variants on the test AUC.

6.2 Model Convergence

Adding DCNv2 came with challenges for model training. During initial training experiments with DCNv2 we observed a large number of runs diverging. To improve model training stability we increased learning rate warm-up from 5% to 50% of training steps. This resolved the instability issues and also significantly boosted the offline relevance gains brought about by adding DCNv2. We also applied batch normalization to the numeric input features as suggested in [36]. Finally, we found that at our number of training steps we were under-fitting. This became clear when we observed that increasing the training steps significantly improved offline relevance metrics. However, increasing the number of training steps was not an option for production due to the decrease in experimentation velocity. As a solution, we found that given the increased warm-up steps, our training was stable enough for higher learning rates. Increasing the learning rate three-fold allowed us to almost completely bridge any relevance metric gaps we found compared to longer training.

We found that optimization needs varied across different models. While Adam was generally effective, models with numerous sparse features required AdaGrad, which significantly impacted their performance. Furthermore, we employed strategies like learning rate warm-up and gradient clipping, especially beneficial for larger batch sizes, to enhance model generalization. We consistently implemented learning rate warm-up for larger batches, increasing the learning rate over a doubled fraction of steps whenever batch size doubled, but not exceeding 60% of the total training steps. By doing so, we improved generalization across various settings and narrowed the gap in generalization at larger batch sizes.

7 CONCLUSION

In this paper, we introduced the *LiRank* framework, encapsulating our experience in developing state-of-the-art models. We discussed various modeling architectures and their combination to create a high-performance model for delivering relevant user recommendations. The insights shared in this paper can benefit practitioners across the industry. *LiRank* has been deployed in multiple domain applications at LinkedIn, resulting in significant production impact.

REFERENCES

[1] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. 2015. Personalizing LinkedIn Feed. In *KDD*.

[2] Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I. Shamir, Rakesh Shivanna, and Qiqi Yan. 2022. On the Factory Floor: ML Engineering for Industrial-Scale Ads Recommendation Models. In *RecSys*.

[3] Limasset Antoine, Rizk Guillaume, Chikhi Rayan, and Peterlongo Pierre. 2017. Fast and Scalable Minimal Perfect Hashing for Massive Key Sets. *SEA* (2017).

[4] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st international workshop on deep learning practice for high-dimensional sparse data*. 1–4.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, and et al. 2016. Wide & Deep Learning for Recommender Systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (2016). https://doi.org/10.1145/2988450.2988454

[6] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2019. Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions. *ArXiv* abs/1909.03276 (2019). https://api.semanticscholar.org/CorpusID:202539143

[7] The Apache Software Foundation. 2020. *Apache Commons Codec*. https://commons.apache.org/proper/commons-codec/ A library of utilities for working with encodings, such as Base64 and Hex.

[8] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211* (2013).

[9] Hui Guan, Andrey Malevich, Jiyan Yang, Jongsoo Park, and Hector Yuen. 2019. Post-training 4-bit quantization on embedding tables. *Published in NeurIPS MLSys Workshop on Systems for ML* (2019).

[10] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *ICML*.

[11] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2021. Soft Calibration Objectives for Neural Networks. In *NEURIPS*.

[12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *ArXiv* abs/1703.04247 (2017). https://api.semanticscholar.org/CorpusID:970388

[13] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. 2019. Applying Deep Learning to Airbnb Search. KDD.

[14] Tongwen Huang, Qingyun She, Zhiqiang Wang, and Junlin Zhang. 2020. GateNet: Gating-Enhanced Deep Network for Click-Through Rate Prediction. *CoRR* abs/2007.03519 (2020).

[15] Shi Jun, Jiang Chengming, Gupta Aman, Zhou Mingzhou, Ouyang Yunbo, Xiao Charles, Song Qingquan, Wu Alice, Wei Haichao, and Huiji Gao. 2022. Generalized Deep Mixed Models. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*.

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2016. Overcoming catastrophic forgetting in neural networks. In *Proceedings of the National Academy of Sciences*.

[17] Lihong Li, Wei Chu, John Langford, Taesup Moon, and Xuanhui Wang. 2011. An Unbiased Offline Evaluation of Contextual Bandit Algorithms with Generalized Linear Models. In *OTEAE*.

[18] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *KDD* (2018). https://api.semanticscholar.org/CorpusID:3930042

[19] 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 *KDD*. 1930–1939.

[20] Kelong Mao, Jieming Zhu, Liangcai Su, Guohao Cai, Yuru Li, and Zhenhua Dong. 2023. FinalMLP: An Enhanced Two-Stream MLP Model for CTR Prediction. In *AAAI Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:257913572

[21] 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 Mikhail Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *ArXiv* abs/1906.00091 (2019). https://api.semanticscholar.org/CorpusID:173990641

[22] 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

[23] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. 2022. PinnerFormer: Sequence Modeling for User Representation at Pinterest. In *KDD*. 3702–3712.

[24] Razvan Pascanu and Yoshua Bengio. 2013. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584* (2013).

[25] Carlos Riquelme, George Tucker, and Jasper Snoek. 2018. Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling. arXiv:stat.ML/1802.09127

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.. In *ICLR*.

[27] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (2020).

[28] 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).

[29] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2018. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2018). https://api.semanticscholar.org/CorpusID:53100214

[30] 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.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[32] Sebastiano Vigna. 2019. *fastutil*. https://github.com/vigna/fastutil A Java library for fast type-specific collections.

[33] Ruoxi Wang, Bin Fu, G. Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. *Proceedings of the ADKDD'17* (2017). https://api.semanticscholar.org/CorpusID:6011288

[34] 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.

[35] Ruoxi Wang, Rakesh Shivanna, Derek Zhiyuan Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. 2020. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. *Proceedings of the Web Conference 2021* (2020). https://api.semanticscholar.org/CorpusID:224845398

[36] Xue Xia, Pong Eksombatchai, Nikil Pancha, Dhruvil Deven Badani, Po-Wei Wang, Neng Gu, Saurabh Vishwas Joshi, Nazanin Farahpour, Zhiyuan Zhang, and Andrew Zhai. 2023. TransAct: Transformer-based Realtime User Action Model for Recommendation at Pinterest. *arXiv preprint arXiv:2306.00248* (2023).

A INFORMATION FOR REPRODUCIBILITY

A.1 Feed Ranking Sparse ID features

The sparse id Feed ranking embedding features consist of (1) Viewer Historical Actor Ids, which were frequently interacted in the past by the viewer, analogous to Viewer-Actor Affinity as in [1], (2) Actor Id, who is the creator of the post, (3) Actor Historical Actor Ids, which are creators who frequently interacted in the past by the creator of the post, (4) Viewer Hashtag Ids, which were frequently interacted in the past by the viewer, (5) Actor Hashtag Ids, which were frequently interacted in the past by the actor of the post and (6) Post Hashtag Ids (e.g. #machinelearning).

We used unlimited dictionary sparse ID features explained in §3.12. We empirically found 30 dimensions to be optimal for the Id embeddings. The sparse id embedding features mentioned above are concatenated with all other dense features and then passed through a multi-layer perception (MLP) consisting of 4 connected layers, each with output dimension of 100.

A.2 Vocabulary Compression for Serving Large Models

The IDs in large personalizing models are often strings and sparse numerical values. If we want to map the unique sparse IDs to embedding index without any collision, then a lookup table is needed which is typically implemented as a hash table (e.g. std::unordered_map in TF). These hash tables grow into several GBs and often take up even more memory than the model parameters. To resolve the serving memory issue, we implemented minimal perfect hashing function (MPHF) [3] in TF Custom Ops, which reduced the memory usage of vocab lookup by 100x. However, we faced a 3x slowdown in training time as the hashing was performed on the fly as part of training. We observed that the maximum value of our IDs could be represented using int32. To compress the vocabulary without degrading the training time, we first hashed the string id into int32 using [7], and then used the map implementation provided by [32] to store the vocabulary. We used a Spark job to perform the hashing and thus were able to avoid training time degradation. The hashing from string to int32 provided us with 93% heap size reduction. We didn't observe significant degradation in engagement metrics because of hashing.

The subsequent effort mentioned in section §3.12 successfully eliminated the static hash table from the model artifact by employing collision-resistant hashing and QR hashing techniques. This removal was achieved without any performance drop, considering both runtime and relevance perspectives.

A.3 External Serving of ID Embeddings vs In-memory Serving

One of the challenges was constrained memory on serving hosts, hindering the deployment of multiple models. To expedite the delivery we initially adopted external serving of model parameters in a key-value store (see Figure 6), partitioning model graphs and pre-computing embeddings for online retrieval. We faced issues with (1) iteration flexibility for ML engineers, who depended on the consumption of ID embeddings, and (2) staleness of pre-computed features pushed daily to the online store. To handle billion-parameter models concurrently from memory, we upgraded hardware and optimized memory consumption by garbage collection tuning, and crafting data representations for model parameters through quantization and ID vocabulary transformation optimized memory usage. As we transitioned to in-memory serving, it yielded enhanced engagement metrics and empowered modelers with reduced operational costs.

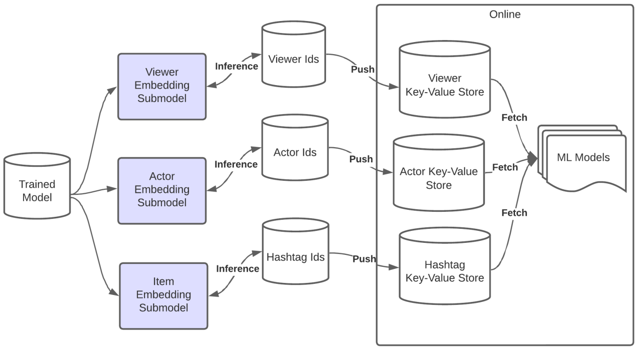

Figure 6: External serving strategy

A.4 4D Model Parallelism

Figure 7 shows an example for three embedding tables. Each embedding table is placed on a GPU, and each GPU's input batch is all-to-all'ed so that every GPU receives the input columns belonging to its embedding table. Each GPU does its local embedding lookup, and the lookups are all-to-all'ed to return the output to the GPU that the input column came from. Other layers with fewer parameters (such as MLP layers) are still processed in a data parallel way since exchanging gradients for these layers is not costly. From our benchmarks, model parallelism reduced training time from 70 hours to 20 hours.

A.5 Experimentation on Sequence Length for User History Models

Here we present study on how history length influences the impact of the Feed ranking model in Table 9. We observe increasing trend of engagement increase as we use longer history of user engagement over sequence architecture described in §3.7.

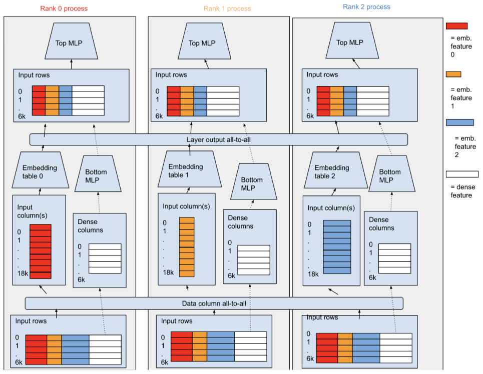

Figure 7: Model parallelism for large embedding tables

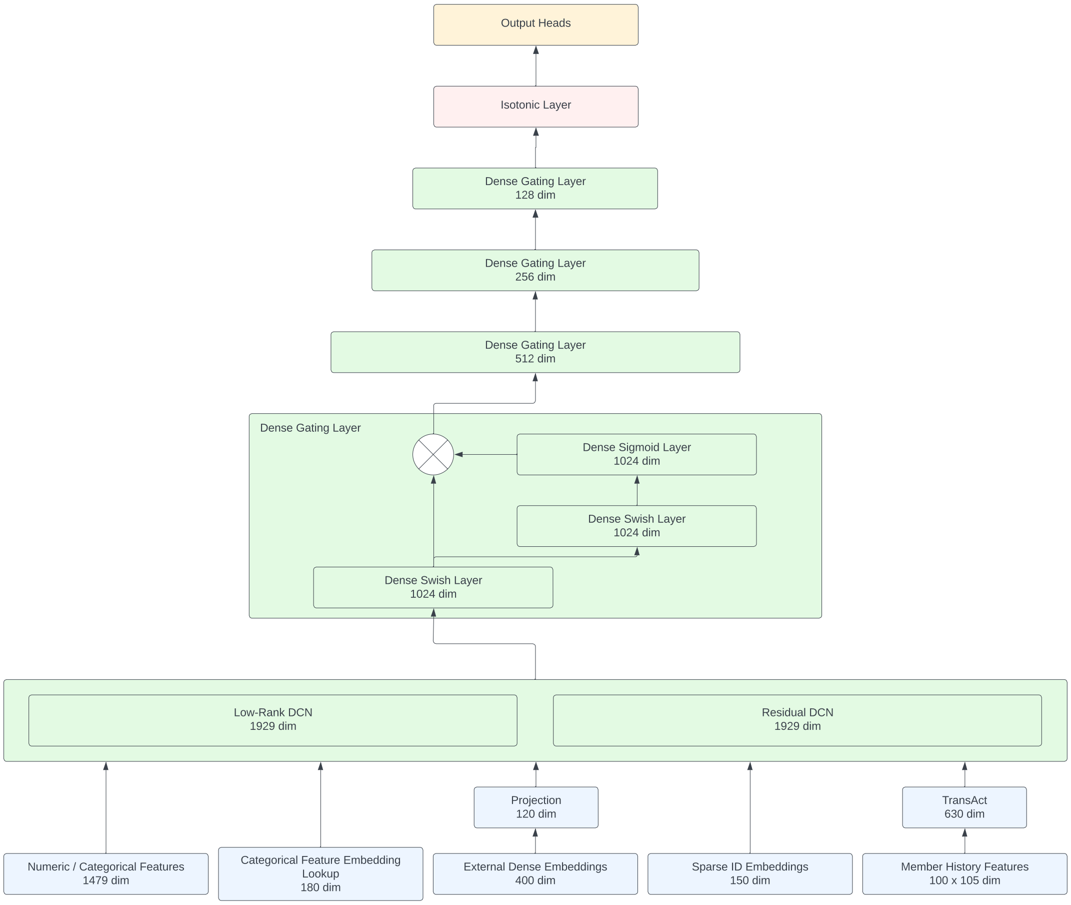

Figure 8: Feed ranking model architecture

Model	Contributions
Baseline	-
+ Member history length 25	+1.31%
+ Member history length 50	+1.57%
+ Member history length 100	+1.66%

Table 9: Offline relevance metrics for the feed from the addition of member history modeling with different sequence lengths.

A.6 Feed Ranking Model Architecture

On the Figure 8 we present Feed Model architecture diagram to provide a flow of the model, and how different parts of the model connected to each other. We found that placement of different modules changes the impact of the techniques significantly.

A.7 Vocabulary Compression

On the Figure 9 we present example diagram of non static vocabulary compression using QR and Murmur hashing. A member ID A in string format like "member:1234," will be mapped with a collision-resistant stateless hashing method (e.g., Murmur hashing) to a space of `int64` or `int32`. The larger space will result in a lower collision rate. In our case, we use `int64`, and then we use `bitcast` to convert this `int64` to two numbers in `int32` space (ranging from 0 to $2^{32} - 1$), B and C which will look from independent sets of QR tables.

A.8 Jobs Recommendations Ranking Model Architecture

As shown in the Figure 10, the jobs recommendation ranking model employs a multi-tasks training framework that unifies Job Search (JS) and Jobs You Might be Interested In (JYMBII) tasks in a single model. The id embedding matrices are added into the bottom layer to be shared by the two tasks, followed by a task-specific 2-layer DCNv2 to learn feature interactions. We conducted various experiments to apply different architectures of feature interactions, and the 2-layer DCN performs best among all. The results on the JYMBII task are demonstrated in the Table **??**.

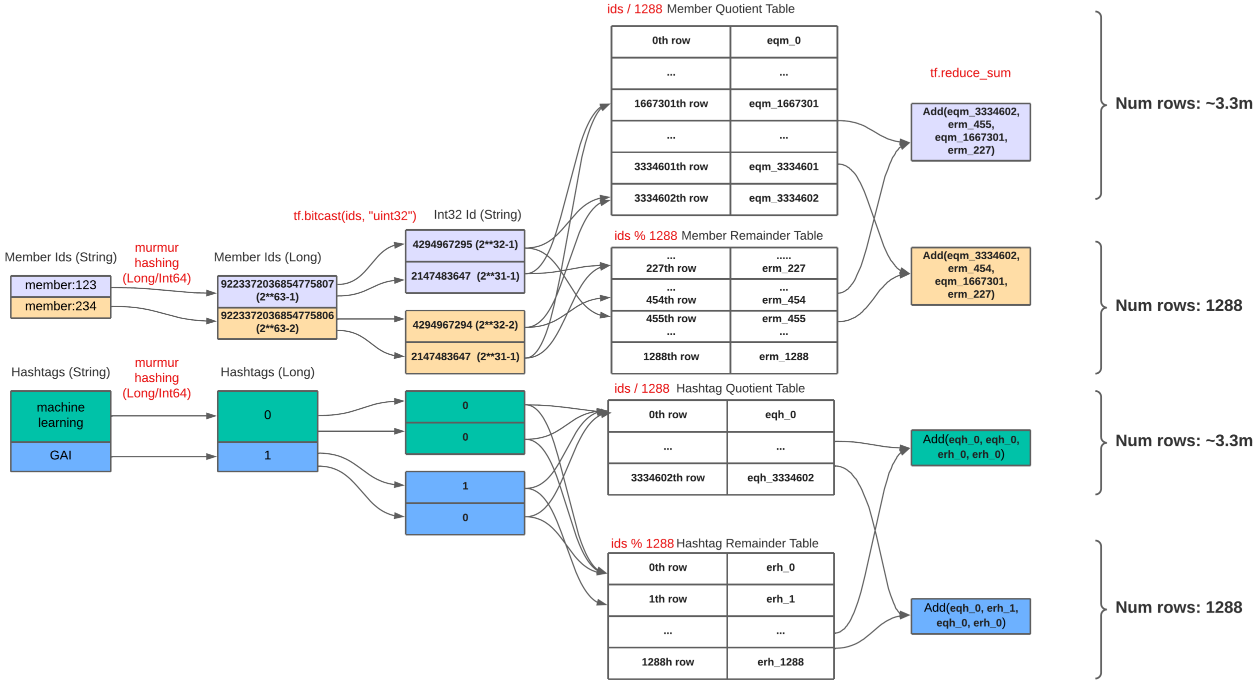

Figure 9: Example of non static vocab hashing paradigm

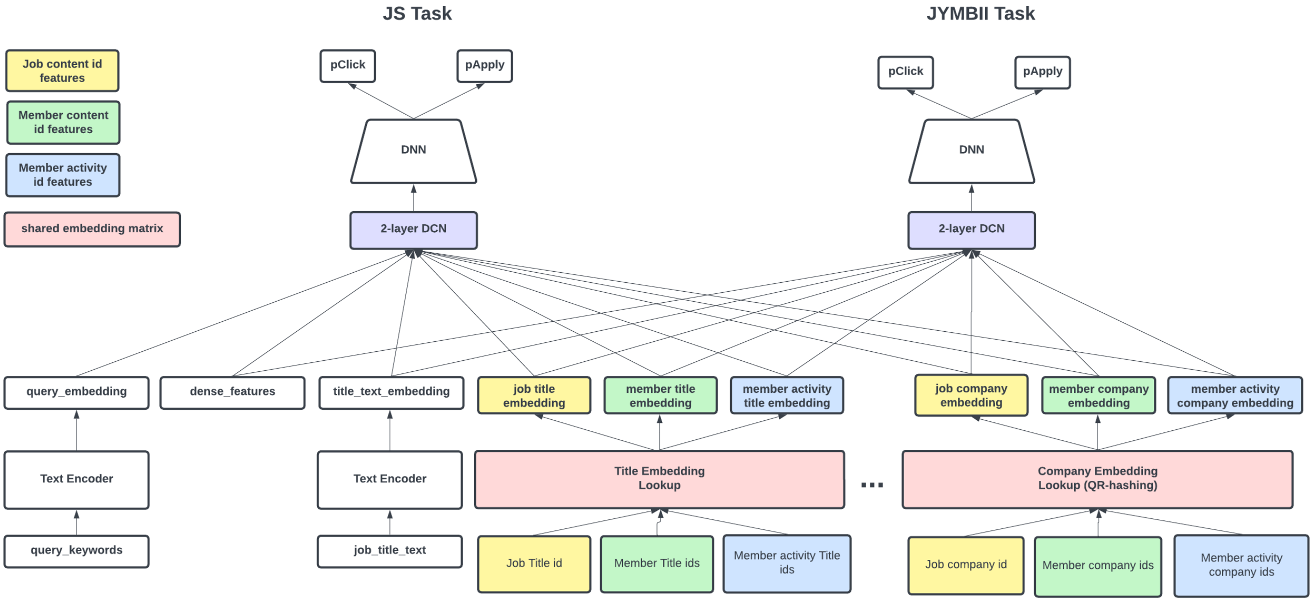

Figure 10: Jobs recommendation ranking model architecture