TwERC: High Performance Ensembled Candidate Generation for Ads Recommendation at Twitter

Vanessa Cai*
Twitter
San Francisco, CA, USA

Pradeep Prabakar*
Twitter
San Francisco, CA, USA

Manuel Serrano Rebuelta
Twitter
New York City, NY, USA

Lucas Rosen
Twitter
New York, NY, USA

Federico Monti
Twitter Cortex
London, UK

Katarzyna Janocha
Twitter Cortex
London, UK

Tomo Lazovich
Twitter Cortex
Boston, MA, USA

Jeetu Raj
Twitter
Seattle, WA, USA

Yedendra Shrinivasan
Twitter Cortex
New York City, NY, USA

Hao Li†
Twitter
Seattle, WA, USA

Thomas Markovich†‡
Twitter Cortex
Boston, MA, USA

ABSTRACT

Recommendation systems are a core feature of social media companies with their uses including recommending organic and promoted contents. Many modern recommendation systems are split into multiple stages - candidate generation and heavy ranking - to balance computational cost against recommendation quality. We focus on the candidate generation phase of a large-scale ads recommendation problem in this paper, and present a machine learning first heterogeneous re-architecture of this stage which we term TwERC. We show that a system that combines a real-time light ranker with sourcing strategies capable of capturing additional information provides validated gains. We present two strategies. The first strategy uses a notion of similarity in the interaction graph, while the second strategy caches previous scores from the ranking stage. The graph based strategy achieves a 4.08% revenue gain and the rankscore based strategy achieves a 1.38% gain. These two strategies have biases that complement both the light ranker and one another. Finally, we describe a set of metrics that we believe are valuable as a means of understanding the complex product trade offs inherent in industrial candidate generation systems.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Personalization**; **Nearest-neighbor search**.

*Both authors contributed equally to this research.
†Equal Contributions
‡Corresponding author: thomasmarkovich@gmail.com

AdKDD, August 7, 2023, Longbeach, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

Candidate Generation, Recommendation Systems, Information Retrieval, Personalization

ACM Reference Format:
Vanessa Cai, Pradeep Prabakar, Manuel Serrano Rebuelta, Lucas Rosen, Federico Monti, Katarzyna Janocha, Tomo Lazovich, Jeetu Raj, Yedendra Shrinivasan, Hao Li, and Thomas Markovich. 2023. TwERC: High Performance Ensembled Candidate Generation for Ads Recommendation at Twitter. In *Proceedings of AdKDD*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern advertising systems are built around recommendation systems, which seek to recommend an advertisement to a user given everything the system knows about the user, similar users, the advertisement, and advertisements like it [5, 9]. Recommendation systems are used to predict a user's expected engagement rate. The predicted engagement rate is usually combined with advertiser specified bid to holistically achieve a desirable balance among users, advertisers, and the platform, so that users are delighted by the ads experience, advertisers are satisfied with the marketing return-on-investment, and the platform can monetize the traffic. The task of ranking all active ads in real-time has become computationally prohibitive as the ads inventories have grown and the model complexity has increased.

To address this issue, modern recommendation systems divide the task into two parts, candidate generation and ranking, as it allows for a more efficient use of computational resources within a tight latency budget, and can lead to better performance compared to a single-stage approach [10]. Candidate generation systems aim to filter the space of all eligible items, which can number in the billions, down to a set of a few thousand likely relevant items for ranking. These methods are often evaluated by recall, which measures the proportion of relevant items that are successfully retrieved. Ranking models then take the relevant items that have been identified by the candidate generation process and use complex models to predict the likelihood of engagement. The goal of ranking models is to sort the relevant items by their predicted utilities (a combination of engagement likelihoods and advertiser specified

bids in the ads recommendation problem). Though ranking models have received extensive study [2, 6, 11, 14, 15], candidate generation systems have received less attention in both the academic literature and industrial applications [12].

In this work, we focus on the candidate generation side of recommendation systems and develop a system called Twitter Ensembled Retrieval of Candidates (TwERC), with a focus on ads applications. In section 2, we present a rebuilt Twitter ads candidate generation stack system that includes a system for combating feedback loops in recommendation systems, a fully counterfactual candidate generation method, a graph similarity-based candidate generation method, and a blender that combines all of these sources together. These pieces combine together to form TwERC. We review both offline and online experiments for TwERC in section 3, and show that this rebuilt system improves the efficiency and effectiveness of the candidate generation process by reducing the number of irrelevant advertisements that are shown to users. Finally, we present our conclusions in section 4.

2 THEORY AND METHODS

After a user opens up or refreshes their home timeline, the page displays a combination of organic recommendations and advertisements. The organic content and advertisements are ranked concurrently, and mixed together at the end of the pipeline with each ad being inserted into the associated advertising slot. The recommendation system built for advertising recommendations contains three major stages: targeting and filtering; early ranking; and heavy ranking and auction. The targeting and filtering applies both hard and soft constraints to the active advertisements, to ensure that we only recommend ads to a user that the advertisers intended the user to see. The set of ads that pass this early filtering stage are then sent to a light ranker. The topK advertisements are then sent to our heavy ranking system, which computes the calibrated probability of engagement for the ad given the optimization objective of that advertisement. In contrast to the light ranker, these heavy ranking models include many heavier-weight features such as richer content representations or graph based user representations [7], and a much more complex architecture [15]. The heavy ranking models output a probability of engagement, pEng, that we then use to compute the rankscore for the second price auction. The rankscore is our best estimate of the overall value or utility of showing a particular ad, and is a function of the advertiser's bid and other factors like predicted engagement rates.

The above system is the product of years of development by many Twitter engineers. While we are always making improvements to the system, we have observed that making improvements to the input feature size and expressiveness of the light ranker is not generally feasible due to the tight latency budget. An alternative way forward to improve the early ranking stage is to perform what we call tail replacement. Tail replacement in this setting specifically means replacing the bottom M% of the topK ads from the light ranker with a different strategy that is complementary and efficient. In this section we explore two separate, but complementary, approaches for tail replacement – rankscore candidate generation and graph based candidate generation, and highlight practical considerations associated with putting these techniques into production. We then discuss the process by which we combine candidates from all three different sources, and finally turn our attention to metrics and figures of merit.

2.1 Rankscore Candidate Generation

Unconstrained Ad Serving The unconstrained ad serving system (UAS) is a counterfactual data collection service that scores all the eligible ad candidates at the request level with a low sample rate. The selected request is duplicated and is then sent to a staging environment to get all ad candidates' bid and predicted engagement rate (positive and negative). In median, UAS scores approximately 50x more than the production scoring volume, granting the capability of observing the quality of ads that might have been filtered out by the early rankers, thus overcoming the infamous selection bias problem in large-scale online interactive/recommendation systems. With the logged bid and predicted engagement rate, we are able to calculate the *rankscore* of all the ad candidates. This data allows us to bypass the common recommendation systems feedback loop, because we are able to rank, and scribe, all relevant ads for a user; rather than a filtered subset. This allows us to train the early ranking system on all data, rather than just its previous outputs. Additionally, we used these data to identify significant headroom (or regret of the current system) by optimizing the top$K$ candidates in the early funnel.

Candidate Generation The counterfactual data is not only used in measuring the regret and/or opportunity size, but also processed to generate user-level top$K$ ad candidates by expected value, through a process of downsampling requests and by removing invalid or incomplete requests. To be more specific, for each user u and eligible ad a we collect $rankscore(u, a, t)$ at time t from UAS with a 3-week lookback window, and calculate the time-aware user-ad quality score $q(u, a)$,

$$q(u, a) = \frac{\sum rankscore(u, a, t) \cdot e^{t-t_0}}{\sum e^{t-t_0}} \tag{1}$$

The quality score is a weighted average at the (u, a) level, with the weight being the elapsed time between the data collection time t and the pipeline running time t_0. Effectively, this self-normalized score gives more weight to the fresher data points. Practically, we observe the time-aware weighting plays a significant role in adapting the volatile nature of the rankscore, and contributes significant incremental business impact on top of a simple average aggregation. Then for each UAS covered user, we generate top$K$ ads with the highest $q(u, a)$ scores. We set up recurring jobs to generate these high value ad candidates every 3 hours and store them in a *Manhattan* dataset (MH) [1], and the MH dataset is consumed by the ad mixer at serving time. Though all the UAS scored ads were eligible as of the data collection time, advertisers sometimes update their budget and targting criteria. To make sure we always fully respect advertisers' targeting criteria and their budget consumption status, we apply an online filter at the serving time too.

2.2 Graph Based Candidate Sourcing

Inspired by previous work at Twitter [7] and elsewhere [12], we choose to construct a heterogeneous graph embedding and use this as the core of our similarity search.

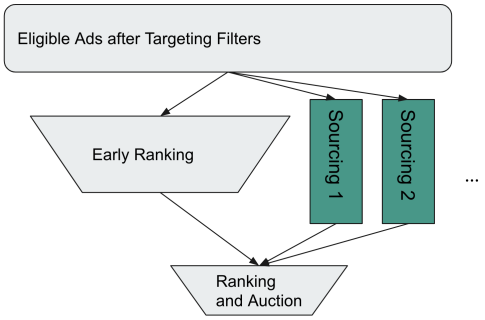

Figure 1: New sourcing components (highlighted) in the serving funnel.

Graph Embeddings For the graph based candidate sourcing strategy, we choose to construct a directed multigraph from the engagements between users and the advertisements they have engaged with. This is done by processing the event level data into a heterogeneous engagement graph with different vertex types (e.g. user, advertisement, advertiser, application, video), and carefully design the edge label definition. While we did not downsample data, we did clean data of all incomplete or otherwise invalid events. Following the recipe of TwHIN [7], we then embed this graph in a lower dimensional space. We generated these TwHIN embeddings using the translation operator with the softmax loss function and dot-product distance function. See [7] for more information. We interpret the distance between a user and an item as the likelihood that user would engage with that item. This strategy produces manifolds in which similar users and similar items are clustered together, and allows us generate candidates through a nearest neighbor search

Embedding Update Cycles The ads engagement graph is constructed by aggregating engagements over a fixed time window. The choice of time window naturally prohibits the representation of interactions as well as vertices which fall outside that window, which make it challenging to keep the graph and associated embeddings up to date.

We choose to address this issue through a warm-starting procedure. Because we have new vertices associated with new users or advertisements, we have to compute new embeddings. Starting the warm-start optimization process with randomly initialized embeddings for new entities would move the already converged vertices, thereby increasing the amount of noise in our embedding updates. For this reason, we freeze the embeddings of the old vertices while the embeddings of the new vertices are computed. One can view this as projecting the new vertices into the existing manifold. Because the old embeddings are frozen, we only need to include edges that include any of the new vertices. We term this set of new edges the Δ edges. This is where the "tic" update cycle stops. These tic embeddings are quick enough that we compute them every twelve hours.

The tic cycle clearly is incapable of capturing the relaxation of the entire graph due to the new interactions and new vertices. To do this, the toc cycle constructs an edge list by combining the Δ edge list with the complete edge list from the previous embedding. With this composite edge list in hand, we then unfreeze the old embeddings and let the entire graph relax until it converges.

Candidate Generation Intuitively, we can retrieve a high-recall set of ad candidates by using a user's TwHIN embedding to query its neighboring promoted tweets using an ANN index [3, 8]. To efficiently retrieve these ads without being constrained by online latency requirements, we built a mapreduce pipeline that precomputes these in batch using the FAISS [8] library for our 130 million heaviest users. We configured FAISS to use the inverted file index + HNSW to improve speed while maintaining high recall. Additionally, we implemented a filtering step to remove ads that are ineligible for a particular user due to targeting criteria or insufficient budget, in order to avoid cache misses during the retrieval stage. We run this pipeline every six hours to both leverage the embedding refresh cycle and avoid storing stale or expired ads.

2.3 Blending

With the initial promising online experiment results from both rankscore candidate generation and graph based candidate sourcing sourcing, we built a blender component in the ads serving early funnel, which dynamically allocate and effectively merging the sourced ads capacities from various sourcing strategies. We use a configuration file ($strategy - percentage$ pairs for each strategy) to specify the capacity allocation among the sourcing strategies, and use it in both offline batch data processing and online serving/merging logic. As an example, in the first blending experiment, we use a 20%-20% blending between the rankscore sourcing and graph based sourcing strategies, with the following key-value configuration {"rankscore": 0.2, "graph": 0.2}. With the specified capacity allocation, we use them to generate a merged *Manhattan* dataset [1], and fill in at most 20% of the candidates with counterfactual-based and graph-based respectively for the full ranking and auction to further decide the final ads impressions (auction winners), meaning we replace up to 40% of the tail.

2.4 Efficiency

Both algorithms that we have proposed can be implemented as batch data pipelines, which feed a distributed key-value store that is utilized for online serving. Because the recommendations are run in batch, the impact to real-time serving is limited to the cost of key-value lookup, which is minimal. The computational resources required are significantly lower than the business impact they provide. Because these pipelines are run in batch, any failures will result in stale recommendations but will not cause serving failures. We have observed an improvement in product metrics if we run the pipelines more frequently, because they provide fresher candidates. We have experimented with a range of refresh-periods and selected the refresh cycle for each algorithm the represents a reasonable tradeoff between computational costs and product metrics. Finally, these pipelines do require maintenance but we have empirically observed that they are quite stable because they are built on top of a robust, distributed, data processing framework.

2.5 Metrics

The choice of metrics is both a deeply interesting question and critical to the success of any project because they will guide the overall progress of research and development. It is important to

find offline metrics that correlate with our various online metrics and product concerns.

Recall The most obvious and easiest to define metric is recall, or hit-rate, defined as $R = \frac{\text{TP}}{\text{TP+FN}}$, where TP is the number of true positives, or hits, and FN is the number of false negatives. We choose to define a hit as an engagement that was correctly identified in the test set. This provides a method to characterize the ability to predict user engagements. While this is valuable, optimizing for pure recall can be sub-optimal because it neglects the role of the auction and the true positive signals can be sparse and has the potential to underweight brand ads.

Auction Recall To address these issues, we introduce the concept of an auction recall. Specifically, we consider true positives to be those ads that win the auction for their slot. Optimizing for auction recall introduces multiple possible issues, including constructing a system that learns the biases and pathological behaviours of the ranking stack that is downstream. While this can lead to short term metric gains both online and offline, it can easily lead to long-term product decay. As a result, we typically examine both auction recall and engagement recall metrics.

Rankscore NCG Rankscore normalized cumulative gain (NCG) compares a given candidate generation algorithm to a hypothetical algorithm that always selects the top ads by *rankscore*. The metric that is computed is the ratio of the sum of the rankscores of the ads selected by the algorithm and the potential rankscore sum that would be found by the hypothetical algorithm. It is defined as:

$$rNCG_m = \frac{\sum_{i=1}^{n} \sum_{j=1}^{min(m,|C_i|)} C_{i,(|C_i|-j+1)}}{\sum_{i=1}^{n} \sum_{j=1}^{min(m,|R_i|)} R_{i,(|R_i|-j+1)}}, \tag{2}$$

where R_i is the set of rankscores of all eligible ads for the ith request after targeting filters are applied, C_i is a the subset of ad candidates selected by the candidate generation algorithm from R_i, n is the number of ad requests, and m is set to match the number of ads that would typically make it to auction. Lower rankscore NCG ratios indicate a larger headroom for showing more highly ranked ads by making improvements to candidate generation.

Inequality Finally, we track the top 1 percent share (T1PS) of the advertisers to evaluate whether our changes are making the overall ads ecosystem more fair. This is important to improving the advertising experience on the Twitter platform because it improves the experience of small and medium sized businesses, which is a long standing product goal for all online advertising platforms. The T1PS coefficient is defined as:

$$T1PS = \frac{\sum_{S_i \geq S_{(0.99|S|)}} S_i}{\sum_i S_i}, \tag{3}$$

where S_i is the number of served ads among all users from advertiser i over a given period. Thus the condition, $S_i \geq S_{(0.99n)}$ restricts to advertisers in the 99th percentile or top 1 percent of advertisers with regards to served ads.

Ads Value It is generally difficult to estimate the average value an advertiser realizes from running an ad campaign on the platform. While we cannot estimate that exact value because we do not have access to the per-advertiser exchange rate for engagements, we can use the average cost per conversion as a natural proxy for that value. Using this intuition, we derive a measure of proxy ads value $\text{AdsValue}(j) = \sum_i \frac{R_i}{C_i} C_{ij}$, where j indicates the experiment bucket, i indicates the campaign index, R_i indicates the revenue for that campaign, C_i indicates the number of conversions for that campaign, and C_{ij} is the number of conversions for the i^{th} campaign in the j^{th} experiment bucket. An increase in ads value corresponds to improvements to our ads ecosystem on the demand side. Tracking the ads value allows us to understand if changes to our recommendation ecosystem are improving the overall long-term value that we provide to our advertisers in an a/b test setting.

Utility Utility is a derived metric that provides a way to estimate the statistical performance of our end to end system. To understand utility, we first have to understand rankscore. Because Twitter is a second price auction, we need to provide our best estimate for the overall value of a particular ad. This is done by combining factors such as the probability of engagement (pEng), probability of negative engagement (pNeg), the advertisers bid. The utility is simply the rankscore with observed values used in place of the probabilities. An increase in utility corresponds to an increase in the quality of our predictions.

3 EXPERIMENTS

Initial investigations found that the ads serving early funnel has significant headroom in the rankscore sum ratio, an ads early funnel metrics indicating the amount of regret we experienced due to filtering. An additional set of experiments found that truncating the tail M% of the early ranker's top$K$ candidates that are sent to full ranking resulted in no observed decrease to net revenue. This indicated an opportunity to improve the filtering phase by experiment with additional candidate generation strategies. We built TwERC, which provides relevant candidates with controlled computational costs to meet this need. To do this, we explored four different strategies that augmented the early ranker. Two of these strategies are deployed in production.

We ran a range of offline and online experiments to evaluate the impact of the two separate sourcing strategies that were constructed. While the experiments presented below are run with each sourcing strategy in isolation, we later on observe that the impact of the two sourcing to be additive due to the construction of the blender.

All online experiments were run using a 2% request sampling strategy, where 2% of the traffic received the treatment. We ran each experiment for a minimum of seven days to account for statistical power and weekly user behavioral trends, and tracked a variety of different metrics to assess improvements to key product metrics. The results presented below are all statistically significant under a Benjamini & Yekutieli correction [4] for multiple comparisons.

3.1 Rankscore Candidate Sourcing

Using the unconstrained ad serving system (see Section 2.1), a counterfactual dataset containing full rankscores of ads, we store high full rankscore ads at the user level, and complement the top$K$ candidate decision previously generated solely by the early ranking model. In practice, we use a trailing 21-day window of the UAS data to generate the user level sourced ads dataset through a time-aware weighted average aggregation which give more weights to more recent rankscore for the user. It is powered by a scheduled batch job that refreshes every 3 hours. At the serving time, we replace at

Objective	RSCS	GBCS	TDGBCS
Obj 1	11.1%	1.1 %	-1.0%
Obj 2	2.3%	5.4%	7.9%
Obj 3	4.7%	2.3 %	10.2%
Obj 4	20.1%	4.5 %	7.2%
Obj 5	15.9%	2.0%	9.9%

Table 1: Offline experiment results from our counterfactual simulator. Here, RSCS is the rank score candidate sourcing strategy, GBCS is the graph based candidate sourcing strategy, and TDGBCS is the time dependent graph based candidate sourcing strategy. All results reflect the auction recall delta between the production baseline and the experimental system.

most M% of the early ranker's tail top$K$ ads by these offline sourced ads, and send the merged candidate ads set to the full ranking.

We prepare the sourced ads at the creative-user level. For the lineitems that are associated with sourced ads, we allow them to bypass the lineitem ranker, but respect all restrictive targeting clauses (also known as targeting filters).

We first tested this strategy using our offline simulator, where we observe improvement, as high as 20% in auction recall across different advertising objectives, as shown in Table 1.

Based on these offline experiments, we ran a range of online experiments with a range of different aggregation approaches. In the final iteration that we shipped to production (Table 2, we observed an 1.38% increase in net revenue, a 4.71% increase in utility per mille impressions, and a 1.05% increase in ads value - all statistically significant with adjusted p-values.

3.2 Graph Based Candidate Sourcing

Previous work using graph embeddings in an offline setting showed promising results on a variety of ranking and candidate generation tasks. Inspired by this work, we looked to explore using these embeddings to generate candidate ads. We began by simulating auction recall after replacing the bottom M% of the tail of the early ranker with nearest neighbor candidates as outlined in section 2.2. In the offline analysis, we found an auction recall improvement across a variety of different objectives as reflected in Table 1.

Motivated by the offline experiments, we hypothesized that by adding graph signals, we would be able to better capture relational information about ads and users that can increase the quality of retrieved ads. To test this hypothesis, we constructed an online experiment where we precompute sourced ads by finding approximate nearest ad neighbors in the graph embedding space for one third of our highly active users offline. These candidates complement the top$K$ candidate decision previously generated solely by the early ranking model. In this experiment, we replaced at most M% of the tail top-ranked ads by the early ranker by these offline sourced ads, and sent the merged candidate ads set to the full ranking. We made no change to the downstream full ranking and auction. As reported in Table 2 we observed statistically significant gains across reported metrics with adjusted p-values. We additionally analyzed this experiment in the context of advertiser inequality, and found a T1PS reduction of 1.2% which was a significant effect. The control bucket result is also consistent with the historical T1PS. When we break down by advertiser type, we see that the small and medium

Method	Revenue	Utility	Ads Value
RSSC	1.38%	4.71 %	1.05 %
GBCS	4.08%	6.42 %	1.26 %

Table 2: Results from the rankscore candidate sourcing online experiment

businesses and mid-market size advertisers experienced the largest decreases in inequality, while direct sale and reseller accounts also experienced moderate decreases. This indicates that our experiment helped the advertisers who were experiencing the largest levels of inequality previously. This experiment is a very concrete example of a case where we can simultaneously increase net revenue and decrease advertiser inequality – two objectives often thought to be in conflict.

3.3 Time Dependent Embeddings

The user embeddings that are used as query vectors generally converge to the average of all of the advertisements that the user engages with. While this provides a reliable way to predict a user's most common commercial interests, it misses interests that may have a temporal nature to them. For example, advertisements for infant goods may become less relevant over time as a parent's child ages. To address this issue, we followed the example of the rankscore temporal discounting in equation (1) and computed time-decayed user embeddings as

$$\vec{u}_{i,t}^{user} = \frac{\sum_{j \in \mathcal{N}_i} e^{\lambda(t_j - t_0)} \vec{v}_j^{tweet}}{\sum_{j \in \mathcal{N}_i} e^{\lambda(t - t_0)}}, \tag{4}$$

where $|\mathcal{N}|$ is the set of the last-N engagements, t_0 is the time the job runs, t_j is the time of the time of the j^{th} interaction, and λ is the decay constant. We implemented this aggregation strategy using a cloud SQL framework. Using this, we ran tail-replacement offline experiments using the same simulator that was used for the other candidate generation experiments, with results reflected in Table 1.

We additionally explored the overlap between candidates using the time-dependent and time-independent. For users with a long history if advertising engagements, we observed little overlap between the two candidate generation strategies. For users with little to no history, however, the overlap increased significantly. This is not unexpected, because the time-dependent vector should be close to the time-independent vector. Finally, we performed a qualitative analysis of the candidates that are generated using these strategies. For users for whom we know their interests, we find this strategy captures emerging interests in some instances, as desired. We are encouraged by these results, as well as the low computational cost to generate these embeddings.

3.4 Embedding Induction

As illustrated in the previous section, using the candidates returned by an ANN-search in the graph embedding space increases the quality of the tweets returned by the early ranker and in turn, the revenue generated by the platform. However, not all users can be served via ANN-search, as users that have never interacted with an ad or an advertiser do not appear in the engagement graph used to train graph embeddings, and thus do not enjoy a representation describing their interests in terms of promoted tweets (a common

situation for new and light users for instance). To enhance the quality of recommended ads to a broader pool of users, we resorted to Graph Learning techniques to extend the coverage of our embeddings. In particular, we implemented an efficient formulation of the Feature Propagation approach described in [13] that propagates information over the arcs of the follow graph. We decided to limit feature propagation to just one hop and infer graph embeddings for missing users U_M as:

$$\boldsymbol{x}_i = \frac{1}{|N^+_{S,T}(i)|} \sum_{j \in N^+_{S,T}(i)} \boldsymbol{x}_j \quad \forall i \in U_M \tag{5}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ and $N^+_{S,T}(i)$ is a randomly sampled set of 100 followings of i that do have an embedding (here sampling is introduced to upper-bound the resources required to process each user). The main intuition is to use the interests of the followings of a target user as a proxy for the interests of the user themselves. With this view in mind, the solution illustrated in (5) can be understood as a form of collaborative filtering where the notion of user-user similarity is defined by follow connections and the averaging mechanism implements a voting system. Empirically, we observed this approach to perform particularly well in offline experiments and be extremely efficient at the same time as equation (5) can be easily implemented with a simple SQL query, which in turn allows to exploit scalable frameworks such as BigQuery for experimentation.

Table 3 shows the results we obtained for users that have and do not have (before diffusion) graph embeddings at experimentation time. Only tweets returned by the ANN-search and that can be recommended to a user as specified by the advertiser's targeting criteria have been considered as possible candidates for a user. A tweet is considered as relevant for a target user if the user has interacted with this in the past.

Method	With	Without
Random ordering	6.5%	10.11%
Inducted graph embeddings	34.38%	42.07%
Graph embeddings	43.53%	-
Light Ranker	49.52%	55.61%

Table 3: Offline results on inducted embeddings for users that with and without a graph embedding (i.e. users that appear in the ad engagement graph).

Inducted embeddings on the follow graph achieve very good performance in this offline setting, obtaining a HR@K equal to $\sim$ 79% of the one showed by the true graph embeddings and maintaining similar performance also on users that do not have a graph embedding at all. At the time of writing, online evaluation of the inducted embeddings is ongoing.

Qualitatively, we observed inducted embeddings to produce good quality results on a few users for which we know their interests, although they might not always capture peculiarities of the users themselves. For instance, users based in a country but that predominantly follow people based abroad tend to get recommended tweets that are consistent with their interests but that are not necessarily targeted to residents of their specific country. This particular phenomenon could be attenuated introducing a weighting scheme on the followings of a target user that takes into account similarities in demographic or potentially engagement. We leave an exploration of this weighting scheme to future work.

4 CONCLUSION

In this work, we described TwERC– a candidate generation system that was designed as part of our ads multistage ranking system. We hypothesized that a heterogeneous candidate generation system can improve performance without substantially increasing computational costs. We demonstrated that an ensemble of different techniques, each with their own bias, are able to provide significant improvements in both offline metrics and product metrics through the use of online A/B tests. As part of these experiments, we also outline a suite of metrics monitor various aspects of the complete ads ecosystem. Finally, we outline multiple exciting new directions to explore to cost-effectively expand this heterogeneous strategy.

REFERENCES

[1] 2014. Manhattan, Our Real-Time, Multi-Tenant Distributed Database for Twitter Scale. https://blog.twitter.com/engineering/en_us/a/2014/manhattan-our-real-time-multi-tenant-distributed-database-for-twitter-scale

[2] Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I Shamir, et al. 2022. On the Factory Floor: ML Engineering for Industrial-Scale Ads Recommendation Models. *arXiv preprint arXiv:2209.05310* (2022).

[3] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. 2008. Nearest neighbor retrieval using distance-based hashing. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 327–336.

[4] Yoav Benjamini and Daniel Yekutieli. 2001. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics* 29, 4 (2001), 1165 – 1188. https://doi.org/10.1214/aos/1013699998

[5] Andrei Z Broder. 2008. Computational advertising and recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*. 1–2.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[7] Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofia Samaniego, Ying Xiao, and Aria Haghighi. 2022. TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM. https://doi.org/10.1145/3534678.3539080

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[9] Seongju Kang, Chaeeun Jeong, and Kwangsue Chung. 2020. Tree-based real-time advertisement recommendation system in online broadcasting. *IEEE Access* 8 (2020), 192693–192702.

[10] Weiwen Liu, Yunjia Xi, Jiarui Qin, Fei Sun, Bo Chen, Weinan Zhang, Rui Zhang, and Ruiming Tang. 2022. Neural re-ranking in multi-stage recommender systems: A review. *arXiv preprint arXiv:2202.06602* (2022).

[11] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[12] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. https://doi.org/10.1145/3394486.3403280

[13] Emanuele Rossi, Henry Kenlay, Maria I Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael Bronstein. 2022. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. *Proceedings of the First Learning on Graphs Conference* (2022).

[14] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[15] 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.