TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation

Ahmed El-Kishky*
Twitter Cortex
Seattle, WA, USA

Thomas Markovich
Twitter Cortex
Boston, MA, USA

Serim Park
Twitter Cortex
San Francisco, CA, USA

Chetan Verma
Twitter Cortex
San Francisco, CA, USA

Baekjin Kim
Twitter
San Francisco, CA, USA

Ramy Eskander
Twitter Cortex
New York, NY, USA

Yury Malkov
Twitter Cortex
San Francisco, CA, USA

Frank Portman
Twitter Cortex
Boston, MA, USA

Sofía Samaniego
Twitter Cortex
San Francisco, CA, USA

Ying Xiao†
Twitter Cortex
San Francisco, CA, USA

Aria Haghighi†
Twitter Cortex
Seattle, WA, USA

ABSTRACT

Social networks, such as Twitter, form a heterogeneous information network (HIN) where nodes represent domain entities (e.g., user, content, advertiser, etc.) and edges represent one of many entity interactions (e.g, a user re-sharing content or "following" another). Interactions from multiple relation types can encode valuable information about social network entities not fully captured by a single relation; for instance, a user's preference for accounts to follow may depend on both user-content engagement interactions and the other users they follow. In this work, we investigate knowledge-graph embeddings for entities in the Twitter HIN (TwHIN); we show that these pretrained representations yield significant offline and online improvement for a diverse range of downstream recommendation and classification tasks: personalized ads rankings, account follow-recommendation, offensive content detection, and search ranking. We discuss design choices and practical challenges of deploying industry-scale HIN embeddings, including compressing them to reduce end-to-end model latency and handling parameter drift across versions.

CCS CONCEPTS

• **Computing methodologies → Learning latent representations**; • **Information systems → Social networks**.

KEYWORDS

heterogeneous information network, social network, recommendation system, embedding, graph embedding, twitter

---

* Corresponding author: aelkishky@twitter.com.
† Equal contribution.

---

1 INTRODUCTION

Twitter is an online social network where users post short messages called Tweets. When users visit Twitter, they can perform a variety of actions – apart from "Favoriting", "Replying" and "Retweeting" tweets (Figure 1); users can also "Follow" other users, search for tweets, click on ads, and open personalized notifications sent to mobile devices. With hundreds of millions of users, and billions of interactions per day, it is a challenging machine learning task to develop holistic (i.e., leverage all our data, independent of modality) and general representations that allow us to understand user preferences and behaviours across the entire platform. Such an understanding is critical in any number of personalized recommendation tasks at Twitter, and for providing compelling user experiences.

Large pretrained embedding tables[1] have become an important tool in recommender systems; they have allowed ML practitioners to understand user behavior. When pretrained embeddings are used as features, they have been shown to improve ranking models [7, 8, 55, 56]. In particular, pretrained user embeddings have been used in a variety of industry recommender systems such as for app and video recommendation for Google Play and Youtube [7, 8], personalized search at AirBnB [16], pin recommendation at Pinterest [31, 55, 56], connecting users based on interest at Etsy [61], stance detection at Twitter [33], and news article recommendation at Yahoo! Japan [30].

The above methods have largely focused on utilizing user-item relations to learn embeddings for an associated recommendation task. For example, PINSAGE [55] exclusively utilizes the "click" or

---

[1]We make the distinction between *pretrained* parameters, which are built independently of a downstream task, and *trainable* embeddings, which are tuned as part of end-to-end task training. We focus on the former case here and do not consider cases of *fine-tuning* where pretrained parameters are used to initialize task parameters, but tuned discriminatively per task.

Figure 1: A mock-up of a Twitter feed. Notice the "Reply", "Retweet" and "Favorite" icons at the bottom of each Tweet.

"repin" actions that Pinterest users take on pins (i.e, content items) to create user embeddings for pin recommendation. While these approaches are successful for the item recommendation task for which they were designed, it may not generalize or be applicable to related recommendation tasks. Returning to the example of Pinterest, there are other entity-relations beyond the user-pin interactions which may be relevant for recommendations. For instance, a user choose to "follow" a variety of other entity-types (another user, board, or topic). A representation which capture all these entity-type relations may be more useful in downstream tasks (such as user-topic or user-board recommendations) than representations learned from user-item interactions alone. These follow relations may also benefit item recommendation, since they are indications of the type of content a user is seeking. In general, leveraging a rich diversity of relations between entities can have many significant advantages to learning representations across many tasks simultaneously:

Data Supplementation: For some tasks, there may simply be fewer data points for training models (e.g., there are generally fewer ad than organic content engagements, or a new product feature may have low density of interactions). In these cases, supplementing low-density relations with information from "denser" relations in the network may improve predictive ability of embeddings derived from the network.

Task Reusability: Often-times, we do not even know all the downstream tasks ahead of time; building 'universal' representations reduces the labour-intensive process of identifying, training, and managing multiple embeddings.

To address these weaknesses, we model multi-type multi-relational networks at Twitter as a heterogeneous information network (HIN) [36, 38], and apply scalable techniques from knowledge graph embeddings (KGE) to embed our heterogeneous networks [1, 22, 42].

Much of the literature in this area either focuses on small-scale embeddings without deploying models to production or industry-scale recommender systems that are trained on simple networks with only a few distinct types of entities and relationships, thereby limiting the embedding's utility to a small number of applications. In this work, we present an end-to-end outline of embedding the **Tw**itter **H**eterogeneous **I**nformation **N**etwork (TwHIN). Our end-to-end system is deployed in production at Twitter, across a variety of product areas; training is scalable, operating on more than 10^9 nodes and 10^{11} edges, and can incorporate many disparate network sources for richer embeddings. TwHIN embeddings capture signals such as social signals (follow-graph), content engagement signals (Tweet, image, and video engagement), advertisement engagements, and others to learn embeddings for users, tweets, and advertisements and other types. We evaluate TwHIN embeddings in online (A/B) tests and offline experiments, demonstrating improvement in multiple tasks.

Compared to previous papers on learning industry-scale embeddings for Web recommender systems, our contributions are:

- We demonstrate simple KGE techniques offer a scalable, flexible scheme for embedding large heterogeneous graphs. This is in contrast to previous industry efforts [55] which require complex alternating CPU-GPU rounds, multiple MapReduce jobs, and custom OpenMP extensions; the KGE embeddings we use need only a single multi-GPU machine.
- We demonstrate how heterogeneous embedding approaches combine disparate network data to effectively leverage rich, abundant unlabeled data sources to learn better representation while simultaneously addressing data sparsity. We show this approach yields gains in multiple downstream recommendation and classification tasks
- We detail practical insights, design considerations and learnings in developing and productionizing a single heterogeneous network embedding for use in a variety of disparate recommender systems within Twitter.

2 RELATED WORKS

Network embedding (a.k.a., graph embedding) techniques have been proposed as a means to represent the nodes of large networks into low-dimensional dense representations [19, 53]. The information contained within these node embeddings has proven useful in a variety of data mining tasks including classification, clustering, link prediction, and recommendation [2, 15, 50]. A family of approaches, starting from DEEPWALK, provide a scalable approach to graph embedding by performing a random walk to create a node sequence, and then applying SkipGram to learn node embeddings [32, 43]. Node2vec extended DeepWalk by applying a biased random walk procedure with a controllable parameter between breadth-first and depth-first sampling strategies [17]. Later methods such as LINE, SDNE, and GraRep incorporate second and higher-order proximity in the node embedding objective [3, 41, 45]. More complex approaches have applied stacked denoising autoencoders to learn node embeddings [4]. Despite the plethora of popular network embedding techniques, these methods largely cater to homogeneous networks – those with a single type of edge relation – with no clear adaptation to type-rich heterogeneous networks.

Generalizing beyond homogeneous networks, heterogeneous information networks have been proposed as a formalism to model rich multi-typed, multi-relational network data [36, 38, 44, 51]. In this setting, one common use-case has been to perform similarity computation between nodes based on structural similarities; several path-based methods have been proposed for this similarity search [35, 39]. Recognizing the utility of HINs in recommendations, there have been many approaches to combining these two ideas. One method addresses the cold-start problem by incorporating heterogeneous network data in social tagging [13]. Another work uses heterogeneous relations in a collaborative filtering social recommendation system [25]. Other approaches have exploited multi-hop meta-paths over HINs to develop collaborative filtering models for personalized recommendation [37]. Additional methods have used these multi-hop meta-paths as latent features within recommender systems [59]. Meta-path similarities were later applied as regularization within matrix factorization recommender systems [57]. Follow-up works have leveraged the rich plethora of entity relationships to perform personalized recommendation [58]. Our approach differs from these methods in that we leverage the plethora of heterogeneous relationships to learn better entity representations (embeddings); these embeddings can then be directly incorporated as dense features in state-of-the-art deep-learning-based recommender models.

Instead of directly operating on the HIN for recommender systems, several papers have investigated learning representations from heterogeneous networks. Several approaches have been developed to learn content-enriched node representations [54, 60]. These methods address data sparsity by leveraging all content associated with nodes. Other methods have been developed to directly embed nodes in a HIN [5, 6, 10, 40, 52]. This work is the closest to our task, however many of these techniques do not easily scale to industry-scale networks such as TwHIN; additionally, several embedding techniques are custom-derived for specific network schema. As an alternative, we apply knowledge graph embedding (KGE) techniques to embed our networks [1, 42, 46]. As KGEs can incorporate mutli-typed nodes and edges, they translate naturally to embedding HINs. Additionally, several frameworks have been developed to scale KGEs to billions of nodes and trillions of edges [22, 62, 63].

3 PRELIMINARIES

Twitter contains a plethora of multi-typed, multi-relational network data. For example, users can engage with other users (i.e., the 'following' relation), which forms the backbone of the social follow-graph. Additionally, users can engage with a variety of non-user entities (e.g., Tweets and advertisements) within the Twitter environment using one of several engagement actions (Favorite, Reply, Retweet, Click). We model these networks as *information networks* [39, 58]:

DEFINITION 1 (INFORMATION NETWORK). *An information network is defined as a directed graph $G = (V, E, \phi, \psi)$ where V is the set of nodes, E is the set of edges, ϕ is an entity-type mapping function ($\phi : V \to \mathcal{T}$) and ψ is an edge-type mapping function ($\psi : E \to \mathcal{R}$). Each entity $v \in V$ belongs to an entity type $\phi(v) \in \mathcal{T}$, and each edge $e \in E$ belongs to a relation type $\psi(e) \in \mathcal{R}$*
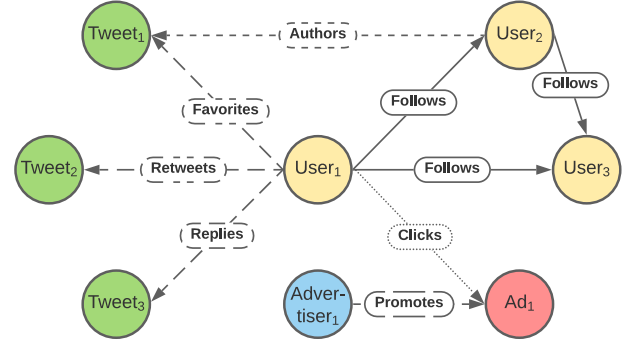

Figure 2: An example heterogeneous information network (HIN) where $|V| = 8$ and $|E| = 9$. There are four entity types ($\mathcal{T}$): 'User', 'Tweet', 'Advertiser', and 'Ad'. There are seven types of relationship ($\mathcal{R}$): 'Follows', 'Authors', 'Favorites', 'Replies', 'Retweets', 'Promotes', and 'Clicks'. See Section 3 for more details.

An information network is a *heterogeneous information network* when $|\mathcal{T}| > 1$ or $|\mathcal{R}| > 1$. For consistency with recommender system terminology, we refer to entities being recommended as *items*. In Figure 2, we give a small example HIN.

Given an input HIN, G and an input dimension d, our goal is to learn embeddings for each entity in G. We define these HIN embeddings as follows:

DEFINITION 2 (HIN EMBEDDINGS). *Given a network $G = (V, E, \phi, \psi)$, the heterogeneous information network embedding uses self-supervised structure prediction tasks in G to map entities in V and relations in $\mathcal{R}$ onto a low-dimensional space $\mathbb{R}^d$, where $d \ll |V|$.*

In particular, our goal is to learn HIN embeddings that provide utility as features in downstream recommendation tasks.

4 TWHIN EMBEDDINGS

In this section, we describe our approach to extracting information from the rich multi-typed, multi-relation Twitter network through large-scale knowledge-graph embedding. We then describe our use of clustering to inductively infer multiple embeddings for each user, and out-of-vocabulary entities such as new tweets without retraining on the new graph. Finally, we describe the overall end-to-end scheme, from the raw data sources to downstream task training.

4.1 HIN embedding approach

We apply knowledge graph embedding to embed the Twitter HIN (TwHIN) [1, 23, 42, 49]. We represent each entity, as well as each edge type in a HIN as an embedding vector (i.e., vector of learnable parameters). We will denote this vector as θ. A triplet of a source entity s, edge type r, and target entity t is scored with a scoring function of the form $f(\theta_s, \theta_r, \theta_t)$. Our training objective seeks to learn θ parameters that maximize a log-likelihood constructed from the scoring function for $(s, r, t) \in E$ and minimize for $(s, r, t) \notin E$.

For model simplicity, we apply translating embeddings (TRANSE) to embed entities and edges in a HIN [1]. For an edge $e = (s, r, t)$, this operation is defined by:

$$f(e) = f(s, r, t) = (\theta_s + \theta_r)^\top \theta_t \tag{1}$$

As seen in Equation 1, TRANSE operates by translating the source entity's embedding with the relation embedding; the translated source and targets embeddings are then scored with a simple scoring function such as a dot product.

We formulate the learning task as an edge (or link) prediction task. We consume the input HIN G as a set of triplets of the form (s, r, t) which represent positive, observed edges. The training objective of the translating embedding model is to find entity representations that are useful for predicting which other entities directly are linked by a specific relation. While a softmax is a natural formulation to predict a linked entity, it is impractical because of the prohibitive cost of computing the normalization over a large vocabulary of entities. As such, following previous methods [14, 27], negative sampling, a simplification of noise-contrastive estimation, is used to learn the parameters θ. We maximize the following negative sampling objective:

$$\arg\max_{\theta} \sum_{e \in G} \left[\log \sigma(f(e)) + \sum_{e' \in N(e)} \log \sigma(-f(e')) \right] \tag{2}$$

where: $N(s, r, t) = \{(s, r, t') : t' \in V\} \cup \{(s', r, t) : s \in V\}$. Equation 2 represents the log-likelihood of predicting a binary "real" or "fake" label for the set of edges in the network (real) along with a set of the "fake" negatively sampled edges. To maximize the objective, we learn θ parameters to differentiate positive edges from negative, unobserved edges. Negative edges are sampled by corrupting positive edges via replacing either the source or target entity in a triple with a negatively sampled entity of the same entity type. As input HINs are very sparse, randomly corrupting an edge in the graph is very likely to be a 'negative' edge absent from the graph. Following previous approaches, negative sampling is performed both uniformly and proportional to entity prevalence in the training HIN [1, 22]. Optimization is performed via Adagrad [11].

4.2 Computational considerations

As an input HIN for Twitter can include more than 10^9 nodes such as Users, Tweets, and other entities, learning an embedding vector for each entity presents both system and GPU memory challenges. We apply the framework PyTorch-Biggraph [22] to address the large memory footprint. This framework randomly partitions using a partition function π each node v into one of P partitions ($\pi(v) \in \{0, \ldots P-1\}$), where P was selected based on the memory available on the server and the GPUs for training. As each node is allocated to a partition, edges $e = (s, r, t)$ are allocated to buckets based on the the source and target nodes (s and t). As such the partitions form P^2 buckets and an edge falls into bucket $B_{\pi(s), \pi(t)}$. Buckets of edges and their associated source and target entities' embedding tables are loaded into memory and embeddings are trained. As such, a maximum of approximately $2V/P$ entities' embeddings are loaded into memory at any point.

Algorithm 1 is then applied to scalably learn θ. The algorithm simply selects a random bucket and loads the associated partitions'

Algorithm 1: HIN Embedding

Input: $G = (V, E, \phi, \psi), epochs, P$
Output: θ (learned embeddings)

1 let θ: initialized embedding vectors entities in V relations in $\mathcal{R}$
2 **for** *each* $\{1 \ldots epochs\}$ **do**
3   **for** *each bucket* $(i, j) : 0 \leq i < P; \ 0 \leq j < P$) **do**
4   load bucket $B_{i,j}$ edges onto memory
5   load $\{\theta_v : \pi(v) = i \lor \pi(v) = j\}$ onto GPU
6   train embeddings on edges using Equation 1
7   **end**
8 **end**

---

embedding tables onto GPU memory. Negative examples for each bucket edge are sampled as described above, but limited only to entities present in the current buckets. Gradients from the edge prediction among negative samples task is backpropagated to learn appropriate embedding vectors.

4.3 HIN at Twitter: TwHIN

When applying Algorithm 1, we must take care to make a crucial distinction between relation types. Within TwHIN we identify relations that are high-coverage in the number of users that participate in the relation, and contrast them to low-coverage relations that are overall sparse. For example, most users follow at least one other user and engage with at least a small number of Tweets. However, many users may not engage with advertisements at all. Recognizing this distinction, we ensure that high-coverage relations are co-embedded with low-coverage relations, but not with other high-coverage relations. This ensures that high-coverage relations benefit entities in low-coverage relations by addressing sparsity, while preserving entities in high-coverage relations from cross-entity type interference.

We form two heterogeneous networks centered around the two high-coverage relations: (1) follow-graph and (2) User-Tweet engagement graph. We refer to these two networks as TwHIN-Follow and TwHIN-Engagement. For each network we augment the high-coverage relation with low-coverage relations with entities such as promoted Tweets (Advertisements), advertisers, and promoted mobile apps. We performed TwHIN embedding on a single computer which allowed for fast prototyping, experimentation, and productionization. We used the Pytorch BigGraph[2] framework to perform embedding at scale [22].

4.4 Inductive multi-modal embeddings

While classical knowledge graph embedding techniques provide a scalable way to embed large HINs, they suffer from two major shortcomings: (1) entites are represented by only a single embedding, which can fail to capture complex behaviour where a user might have disparate multi-modal interests (2) entity embeddings are *transductive* and only defined for entities present at training time and retraining is required on the full graph to support any novel out-of-vocabulary nodes. This latter shortcoming is significant for application to TwHIN since the set of Tweets and Users change rapidly over time.

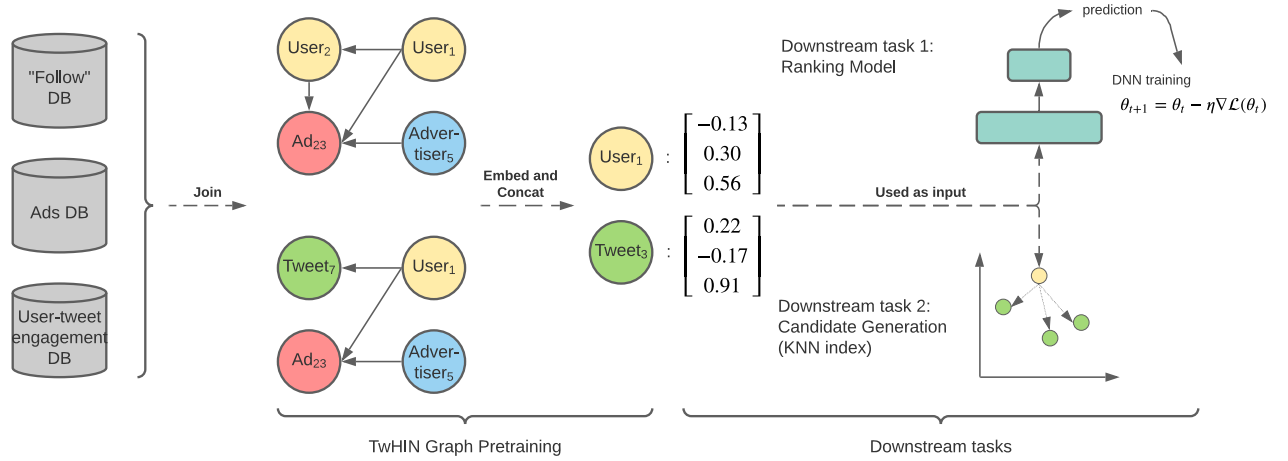[2]https://github.com/facebookresearch/PyTorch-BigGraph

Figure 3: The end-to-end framework aggregates disparate network data to construct TwHIN, joint-embedding is performed and embeddings are consumed in downstream tasks and ML models.

We address both of these short-comings by introducing a fast post-processing step that can represent TwHIN entities as mixture over multiple embeddings. This technique is flexible enough to *inductively* embed new, out-of-vocabulary, nodes such as Tweets. To create these embeddings for a node type we (1) cluster existing unimodal embeddings (2) compute multiple embeddings for a node by aggregating the most engaged-with clusters for a node.

To illustrate this approach, let us focus on transforming User embeddings into multi-modal mixtures of embeddings. We first consider set T of non-user entities that have some engagement with users; we think of T as the set of "targets" and will seek embeddings for users that summarize a user's engagement with elements of T. We first take the set of targets and perform k-means clustering [34] to create a set of clusters C over T. We then associate each user u_i with a probability distribution over elements of C proportional to the amount of engagement the user has with each cluster:

$$P(c|u_i) = \frac{\text{count}(u_i, c)}{\sum\limits_{c' \in \mathcal{M}(u_i)} \text{count}(u_i, c')} \qquad (3)$$

Where, $\text{count}(u_i, c)$ is the number of times u_i engages with a target in cluster c and, for computational efficiency, we take $\mathcal{M}(u_i)$ to be u_i's top m most engaged clusters. We normalize these cluster engagements to obtain a proper cluster-engagement distribution. In addition to each user's TwHIN entity embedding, we can now represent each user with a mixture distribution over their top m most engaged cluster with each cluster represented by a dense vector of its centroid or medoid. This multi-modal representation addresses both of the short-comings since different clusters of target entities may better capture complex behavior and they can also be generated for entities that were unseen during training.

The multi-modal embeddings here have similar motivations as those in PINNERSAGE [31], but have some key differences. In that work a user's target engagements are clustered and individual items capturing disparate interests are used to represent the user for recommendation retrieval. By contrast, in the approach here, we are clustering the entire universe of items (Tweets and other non-user

entities) and representing users according to which of these global clusters their engagements coincide. So rather than run clustering for each user on just their engagement, we instead run a single large clustering job over non-user entities.

5 TWHIN FOR RECOMMENDATION

In Figure 3, we show our end-to-end framework including collecting disparate data sources to organize TwHIN, learning entity embeddings through a self-supervised KGE objective, and finally using the TwHIN embeddings in downstream tasks. In this section we discuss using TwHIN embeddings for two families of tasks: (1) candidate generation and (2) as features in deep-learning models for recommendation and prediction.

5.1 TwHIN candidate generation

Candidate generation is the first step in most recommender systems; the aim is to retrieve a user-specific high-recall set of relevant candidates using a light-weight strategy. Within Twitter, we use an approximate nearest neighbor (ANN) system that indexes items to be suggested such as Users to Follow or Tweets to engage with. Two internal systems use HNSW [26] or FAISS [21] with Product Quantization [20] to index items and retrieve candidates.

We then use an entity's TwHIN embedding to query candidate entities of any type (assuming a distinct index per entity-type). However, when indexing a large number of items such as users or Tweets, many of the retrieved items may be very similar. These are not desirable as users get diminishing value from being presented near duplicate items. To address this, we use multi-modal embeddings (see Section 4.4) to generate diverse candidates. Given an entity's multimodal representation as a *mixture over multiple embeddings* with non-negative mixture coefficients normalized to one, we can query candidates from each vector in its mixture representation and select a number of candidates proportional to the query vector's mixture weights. This adds diversity as candidates are explicitly queried from different areas of the embedding space.

5.2 TwHIN for ranking and prediction

Many supervised machine learning models at Twitter employ pretrained TwHIN embeddings as features. These models are used in a variety of tasks from recommendations ranking, to content classification, and other predictive tasks. Many standard deep neural network (DNN) models have been applied to predictive modeling [7, 47, 48]. Recommendation ranking models typically take as input a set of user and contextual information, and output is a ranked list of items from the candidate generation step based on objectives such as engagement or purchase probability. Predictive classification models take similar features and predict a classification label such as topic or other content classification.

At Twitter, predictive models take in users and/or items as described by many continuous and categorical features. Categorical features are then associated with an emebdding vector via a look-up table; these embedding vectors are task-specific and learned while training each DNN model. Continuous features are then appended to these embeddings and the concatenated feature-set is fed into a DNN (e.g., MLP) and trained on a task-specific objective. To incorporate TwHIN embeddings, we employ a look-up table to map an entity id to its associated pretrained TwHIN embedding. However, unlike with other categorical features, these pretrained embeddings are frozen and not trained with the unfrozen embeddings.

6 EXPERIMENTS & RESULTS

We experimentally demonstrate the generality and utility of TwHIN embeddings via online and offline experimentation on several Twitter internal ML models and tasks.

6.1 Candidate generation

Our first family of tasks are *candidate generation* tasks; these are tasks that select a high-recall pool of items that are then ranked by more complex downstream ranking models. We demonstrate offline and online gains in using TwHIN embeddings on a "Who to Follow" task.

Who to Follow Suggestions: We describe results from leveraging TwHIN embeddings for the *Who to Follow* [12, 18] user recommendation task which suggests Twitter accounts for a user to follow. We utilize TwHIN user embeddings as a query to retrieve highly followed users via approximate nearest neighbor search. We compare using a single user embedding for querying to using multi-modal "mixture of embeddings" as described in Section 4.4 and Section 5.1.

Approach	R@10	R@20	R@50
Unimodal	0.58%	1.02%	2.06%
Mixture	3.70%	5.53%	8.79%

Table 1: Comparing candidate generation using a single TwHIN embedding vs a mixture of embeddings with multiquerying.

Table 1 compares the performance of multi-modal mixtures of TwHIN user embeddings vs unimodal user embeddings. Across all thresholds (recall at 10, 20, and 50), multi-modal mixtures with multi-querying significantly outperform single unimodal representations. This confirms the hypothesis that mixtures of embeddings

better model users and their multiple interests. Explicitly querying from different parts of the embedding space consistently yielded over 300% improvement in recall over unimodal representation querying.

6.2 Recommendation and prediction

We incorporate TwHIN embeddings in several supervised recommendation and prediction tasks: (1) *Predictive Advertisement Ranking* (2) *Search Ranking*, and (3) *Offensive Content Detection*

Predictive Advertisement Ranking: Following procedures in Section 5, we add TwHIN embeddings for different entities to several ads recommendation models [29]. We are unable to disclose specific details of each model, or the timeline in which they were deployed. As such, we simply refer to these models as Ads_1, Ads_2 and Ads_3; each targets a different ads target objective, but does share some, but not all, hand-crafted features. In particular, each model has many features about users and their interactions with ads.

We evaluate our model quality using Relative Cross Entropy (RCE). This metric measures how much better an ads engagement predictor is compared to a naive prediction using the prior of each label. This is computed as follows:

$$\text{RCE} = 100 \times \frac{\text{Reference Cross Entropy} - \text{Cross Entropy}}{\text{Reference Cross Entropy}} \quad (4)$$

where the reference cross entropy is that of the prior, and the cross entropy term is that of the treatment.

During online A/B experiments, where the approaches were tested on live traffic, we computed the pooled RCE (using both control and treatment traffic) and noticed a significant improvement when adding TwHIN embeddings over the baseline model. Adding TwHIN embeddings yielded an average **2.38** RCE gain over the baseline resulting in a 10.3% cost-per-conversion reduction in the new production model. These results allowed us to deploy TwHIN embeddings for several additional advertisement models which all demonstrated online improvement. To further validate our hypothesis as to the importance of heterogeneity in TwHIN, we perform offline entity ablation studies for a set of advertisement models. Note that in some of our experiments, we observe that corresponding online RCE increase is a lot more significant than what is measured offline due to differences between offline and online environment. Notwithstanding, the offline results are directionally monotonic with those measured online. In particular, any offline results should be used to compare utilizing different entity embeddings within the same model.

Model	Baseline	U	U+A	U+T	U+A+T
Ads_1	21.23	21.32	21.43	21.46	**21.48**
Ads_2	13.53	13.61	13.54	**13.63**	13.59
Ads_3	17.11	17.26	17.27	**17.27**	17.26

Table 2: Offline RCE for ads models with feature ablation. We investigate performance when using TwHIN embeddings for User (U), Advertiser (A), and Target entity (T) such as app to install, video to watch, or advertisement to click.

As a general trend, we notice the most improvement arises from user embeddings. However, we do see further improvements when adding entity embeddings for Advertiser and the exact target (e.g., promoted app for installation, promoted video, or promoted Tweet). This supports our intuition that leveraging multi-type embeddings can improve downstream predictive models.

We also ran experiments (not shown) where we directly embedded low-coverage relationships without augmenting with denser relationships. This yielded much lower improvements, and sometimes even model degradation. This further supports our claim that denser relations can supplement less dense relations.

Search Ranking: We investigate using TwHIN embeddings to improving personalized search ranking. Personalized search ranking consists of a search session for a user where they provide an input query and the system ranks a set of Tweets based on engagement. Our baseline ranking system takes as input a large set of hand-crafted features that represent the underlying user, query and candidate Tweets. In addition, the input features also include the outputs of an mBERT [9] variant fine-tuned on in-domain query-Tweet engagements to encode the textual content of queries and Tweets. The hand-crafted and contextual features are fed into an MLP, where the training objective is to predict whether a Tweet triggers searcher engagement or not.

We augment this baseline model with three TwHIN embeddings: User embeddings from both follow-base (U_f) and engagement-base (U_e), as well as Author embeddings (A).

Metric	Baseline	$+U_f$	$+U_e$	$+A$	$+U_f + U_e$	$+U_f + U_e + A$
MAP	55.7	56.2	56.6	55.8	56.6	**57.0**
ROC	57.9	58.6	59.0	57.9	59.0	**59.6**

Table 3: Search engagement-based ranking with TwHIN embeddings: TwHIN user embeddings, both follow-base (U_f) and engagement-base (U_e), and TwHIN author embeddings (A)

We train our models on search-engagement data. We fine-tune the hyper-parameters on search sessions from a held-out validation day and report ranking performance in Table 3 for both MAP and averaged ROC using search sessions from a held-out test day.

As seen in Table 3, combining the three types of TwHIN embeddings as additional inputs to the baseline system yields the best ranking performance with relative error reductions of 2.8% in MAP and 4.0% in averaged ROC. Note also that TwHIN user embeddings (U_f and U_e), either independently or in conjunction, yield performance gains, unlike TwHIN author embeddings (A), which only help when combined with user embeddings.

Detecting Offensive Content: We evaluate TwHIN embeddings for the task of predicting whether a Tweet is offensive or promotes abusive behavior.[3] While the definition of this problem is purely concerned with the Tweet content, we hypothesize that a key element of interpreting intent of a Tweet is understanding the social context and community of the Tweet author. These experiments

---

[3]See help.twitter.com/en/rules-and-policies/abusive-behavior for a fuller discussion of what is considered abusive behavior in Tweets.

	RoBERTa	BERTweet	+TwHIN-Author
Collection$_1$	0.4123	0.4692	**0.5161**
Collection$_2$	0.688	**0.7274**	0.7174

Table 4: PR-AUC results for detecting offensive content. We compare the performance of content-based models to leveraging a TwHIN author embedding in addition to content.

are purely academic, and TwHIN is not currently being applied to detecting offensive content at Twitter.

For our experimental purposes, we construct a baseline approach that fine-tunes a large-scale language model for offensive content detection using linear probing and binary categorical loss; we compare the performance of RoBERTa [24] and BERTweet [28] language model, the latter of which has been pretrained on Twitter-domain data. We evaluate on two collections of tweets where some tweets have been labeled "offensive" or violating guidelines. The baselines leverage pretrained language models to embed the textual content. We supplement the stronger baseline by concatenating TwHIN author embedding to the language model content embedding; linear probing is used for fine-tuning.

Results in Table 4 confirm our hypothesis showing that adding TwHIN embedding increases PR-AUC by an average relative gain of 9.09% on Collection$_1$ with neutral results on Collection$_2$, likely stemming from Collection$_2$ containing a very high proportion of offensive tweets. This experimental result confirms unrelated relationships (e.g., Follows and Tweet engagements) can be used to pretrain user embeddings that can improve unrelated predictive tasks such as offensive content or abuse detection validating our claim on the generality of our TwHIN embeddings. While TwHIN was constructed without any data from this task, the learned embeddings were able to significantly improve performance on this task.

7 PRACTICAL CONSIDERATIONS
We discuss design decisions made in productionizing TwHIN with regards to (1) latency concerns and (2) mitigating technical debt by minimizing parameter drift.

7.1 Compression for low latency
Performance in downstream ranking task often improves substantially as we increase the embedding dimension. Since, TwHIN embeddings are designed to be used in many latency-critical online recommender systems such as advertisement ranking and content ranking, we apply a simple and effective lossy data-based compression scheme via product quantization [20]:

(1) After generating the embeddings, we train a product quantization scheme (we use the FAISS package[4] [21]), and export the product quantization codebook (centers).
(2) Upon encountering a new downstream task, we make the codebook available to the training process, decoding the compressed embeddings using a codebook lookup.
(3) At inference time, we once again perform a codebook lookup.

---

[4]https://github.com/facebookresearch/faiss

(a) Recall@10 for k-nearest neighbors with varying levels of compression for TwHIN.

(b) Ave. ROC for search ranking model lwith varying levels of compression for TwHIN.

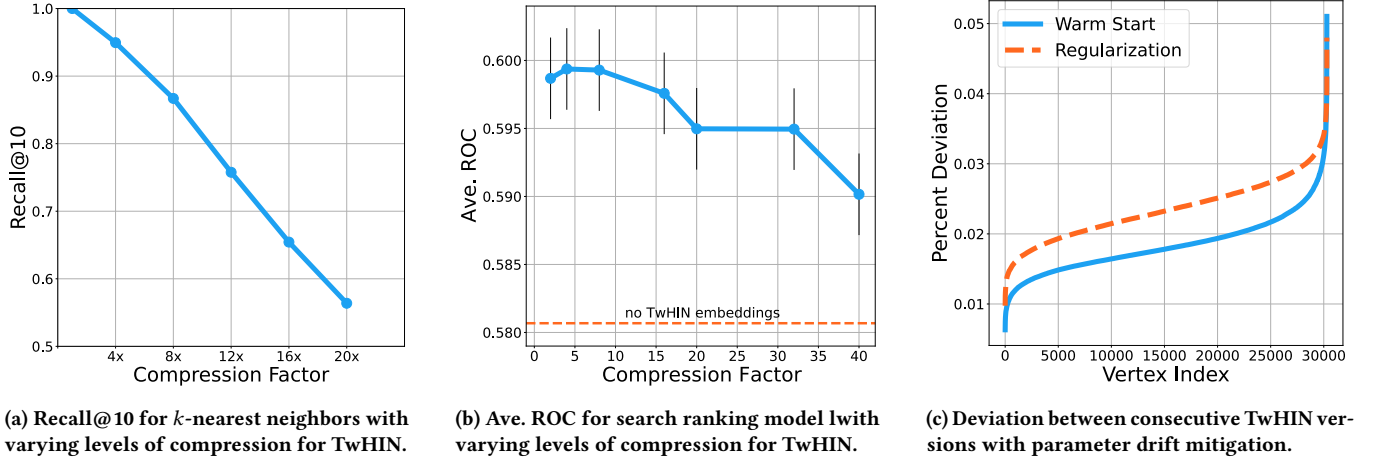(c) Deviation between consecutive TwHIN versions with parameter drift mitigation.

Figure 4: Experiments exploring effects of compression on performance, and parameter drift mitigation strategies on drift.

This scheme reduces the input size and network IO significantly, yields essentially identical downstream model performance, and introduces negligible latency. In Figure 4a, we consider the tradeoff between compression factor and the accuracy of decompressed embeddings in a k-nearest neighbors task. Even at high compression levels (20×), we still maintain reasonable accuracy in the k-nearest neighbor task. Similarly, the effects of compression on the supervised search ranking task from Section 6.2, show negligible effect on model performance even up to 30× compression. These results motivate our approach to utilizing product quantization with model-side codebook decompression in our latency-critical recommendation tasks.

7.2 Addressing parameter drift across versions

Since the underlying information network contains user behaviors (e.g, follow or engagement actions) that evolve over time, TwHIN embeddings must be updated regularly to accurately represent entities. However, in doing so, we do not want the embeddings to drift too much from their current values, since we do want to simultaneously re-train all the downstream models.

Naively re-training the TwHIN embedding will lead to very large drifts caused by random initialization and stochasticity of optimization. In response, we have tried two natural approaches to achieving stability for embeddings: warm start and regularization. In the warm start approach, we simply initialize embeddings in the new version with the prior version's values. When we encounter new vertices that weren't previously seen, we either randomly assign them vectors, or initialize the vectors according to:

$$\theta_v = \frac{1}{|\mathcal{N}_v|} \sum_{v' \in \mathcal{N}_v} \theta_{v'} + \theta_{\psi(v,v')} \quad (5)$$

where θ_v is the embedding vector for node v, $\mathcal{N}_v$ is the graph neighborhood around v, and $\theta_{\psi v,v'}$, the relationship vector learned between vertices v and v'.

Alternatively, adding regularization is a more principled way of addressing this issue, allowing us to directly penalize divergence

from a past version. The simplest way is to apply L2 regularization to the previous embedding: $\alpha \left\| \sum_v \left(\theta_v - \theta_v^{prev} \right) \right\|_2^2$. Although this method is notionally simple, it presents the disadvantage of doubling the memory requirements, since we must also use θ^{prev}.

We evaluate these two methods in terms of (1) parameter changes in L2 distance (2) the effect on downstream tasks. To assess parameter changes in L2 distance, we first generated a TwHIN embedding while optimizing for 30 epochs. Afterwards, we re-trained for 5 epochs, separately applying warm-starting and L2 regularization. In Figure 4c, warm-starting is better at minimizing deviations except in instances where the vertices have very high degree. Intuitively, this makes sense because the high-degree nodes are able to 'overwhelm' the regularizer with their loss. Even still, a maximally 0.05% deviation is more than sufficient to fulfill our stability requirements.

Metric	Control	Warm Start	Regularizer
R@10	21.71	**21.75**	21.74
MRR	0.0635	0.0635	**0.0638**

Table 5: Performance of various parameter-drift minimization strategies on Who to Follow task.

To evaluate the effect on downstream tasks, in Table 5, we present a comparison on the Who to Follow task (Section 5.1). As we see in the above results, both warm start and regularization preserve stability in this downstream task. In practice, we have internally chosen to update TwHIN versions using the warm start strategy due to its space efficiency and simplicity.

8 CONCLUSION

In this work, we describe TwHIN, Twitter's in-house joint embedding of multi-type, multi-relation networks with in-total over a billion nodes and hundreds of billions of edges. We posit that joint embeddings of heterogeneous nodes and relations is a superior paradigm over single relation embeddings to alleviate data sparsity issues and improve generalizability. We demonstrate that simple, knowledge graph embedding techniques are suitable for large-scale

heterogeneous social graph embeddings due to scalability and ease at incorporating heterogeneous relations. We deployed TwHIN at Twitter and evaluated the learned embeddings on a multitude of candidate generation and personalized ranking tasks. Offline and online A/B experiments demonstrate substantial improvements demonstrating the generality and utility of TwHIN embeddings. Finally, we detail many "tricks-of-the-trade" to effectively implement, deploy, and leverage large scale heterogeneous graph embeddings for many latency-critical recommendation and prediction tasks.

REFERENCES

[1] A. Bordes, N. Usunier, A. Garcia-Duran, J Weston, and O. Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *NeurIPS* 26 (2013).
[2] H. Cai, V. Zheng, and K. Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE* 30, 9 (2018), 1616–1637.
[3] S. Cao, W. Lu, and Q. Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.
[4] S. Cao, W. Lu, and Q. Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*.
[5] S. Chang, W. Han, J. Tang, Q. Qi, C. Aggarwal, and T. Huang. 2015. Heterogeneous network embedding via deep architectures. In *SIGKDD*. 119–128.
[6] T. Chen and Y. Sun. 2017. Task-guided and path-augmented heterogeneous network embedding for author identification. In *WSDM*. 295–304.
[7] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. 2016. Wide & deep learning for recommender systems. In *DLRS*. 7–10.
[8] P. Covington, J. Adams, and E. Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys*. 191–198.
[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805* (2018).
[10] Y. Dong, N. Chawla, and A. Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*. 135–144.
[11] J. Duchi, E. Hazan, and Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12, 7 (2011).
[12] Ahmed El-Kishky, Thomas Markovich, Kenny Leung, Frank Portman, Aria Haghighi, and Ying Xiao. 2022. kNN-Embed: Locally Smoothed Embedding Mixtures For Multi-interest Candidate Retrieval. *arXiv preprint arXiv:2205.06205* (2022).
[13] W. Feng and J. Wang. 2012. Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In *SIGKDD*. 1276–1284.
[14] Y. Goldberg and O. Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv:1402.3722* (2014).
[15] P. Goyal and E. Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.
[16] M. Grbovic and H. Cheng. 2018. Real-time personalization using embeddings for search ranking at airbnb. In *SIGKDD*. 311–320.
[17] A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.
[18] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. 2013. Wtf: The who to follow service at twitter. In *WWW*. 505–514.
[19] P. Hoff, A. Raftery, and M. Handcock. 2002. Latent space approaches to social network analysis. *JASA* 97, 460 (2002), 1090–1098.
[20] H. Jegou, M. Douze, and C. Schmid. 2010. Product quantization for nearest neighbor search. *IEEE TPAMI* 33, 1 (2010), 117–128.
[21] J. Johnson, Ma. Douze, and H. Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).
[22] Ad. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).
[23] Y. Lin, Z.n Liu, M. Sun, Y. Liu, and X. Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*.
[24] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[25] C. Luo, W. Pang, Z. Wang, and C. Lin. 2014. Hete-cf: Social-based collaborative filtering recommendation using heterogeneous relations. In *ICDM*. 917–922.
[26] Y. Malkov and D. Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *PAMI* (2018).
[27] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. *NeurIPS* 26 (2013).
[28] D. Nguyen, T. Vu, and A. Nguyen. 2020. BERTweet: A pre-trained language model for English Tweets. *arXiv preprint arXiv:2005.10200* (2020).
[29] C. O'Brien, K. Liu, J. Neufeld, R. Barreto, and J. Hunt. 2021. An Analysis Of Entire Space Multi-Task Models For Post-Click Conversion Prediction. In *RecSys*.
[30] S. Okura, Y. Tagami, S. Ono, and A. Tajima. 2017. Embedding-based news recommendation for millions of users. In *SIGKDD*. 1933–1942.
[31] A. Pal, C. Eksombatchai, Y. Zhou, B. Zhao, C. Rosenberg, and J. Leskovec. 2020. PinnerSage: multi-modal user embedding framework for recommendations at pinterest. In *SIGKDD*. 2311–2320.
[32] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.
[33] J. Pougué-Biyong, A. Gupta, A Haghighi, and A El-Kishky. 2022. Learning Stance Embeddings from Signed Social Graphs. arXiv:2201.11675 [cs.SI]
[34] D. Sculley. 2010. Web-scale k-means clustering. In *WWW*. 1177–1178.
[35] C. Shi, X. Kong, Y. Huang, Y. Philip, and B. Wu. 2014. Hetesim: A general framework for relevance measure in heterogeneous networks. *TKDE* (2014).
[36] C. Shi, Y. Li, J. Zhang, Y. Sun, and Y. Philip. 2016. A survey of heterogeneous information network analysis. *TKDE* 29, 1 (2016), 17–37.
[37] C. Shi, Z. Zhang, P. Luo, P. Yu, Y. Yue, and B. Wu. 2015. Semantic path based personalized recommendation on weighted heterogeneous information networks. In *CIKM*. 453–462.
[38] Y. Sun and J. Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter* (2013).
[39] Y. Sun, J. Han, X. Yan, P. Yu, and T. Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB* (2011).
[40] J. Tang, M. Qu, and Q. Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *SIGKDD*. 1165–1174.
[41] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[42] T. Trouillon, J Welbl, S. Riedel, É. Gaussier, and G. Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*. PMLR, 2071–2080.
[43] C. Tu, W. Zhang, Z. Liu, M. Sun, et al. 2016. Max-margin deepwalk: Discriminative learning of network representation.. In *IJCAI*, Vol. 2016. 3889–3895.
[44] C. Wang, Y. Song, A. El-Kishky, D. Roth, M. Zhang, and J. Han. 2015. Incorporating world knowledge to document clustering via heterogeneous information networks. In *SIGKDD*. 1215–1224.
[45] D. Wang, P. Cui, and W. Zhu. 2016. Structural deep network embedding. In *SIGKDD*. 1225–1234.
[46] Q. Wang, Z. Mao, B. Wang, and L. Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *TKDE* 29, 12 (2017), 2724–2743.
[47] R. Wang, B Fu, G Fu, and M. Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD*. 1–7.
[48] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi. 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *WWW*. 1785–1797.
[49] Z. Wang, J. Zhang, J. Feng, and Z. Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, Vol. 28.
[50] X. Wei, L. Xu, B. Cao, and P. Yu. 2017. Cross view link prediction by learning noise-resilient representation consensus. In *WWW*. 1611–1619.
[51] D. Xin, A. El-Kishky, D. Liao, B. Norick, and J. Han. 2018. Active learning on heterogeneous information networks: A multi-armed bandit approach. In *ICDM*.
[52] L. Xu, X. Wei, J. Cao, and P. Yu. 2017. Embedding of embedding (EOE) joint embedding for coupled heterogeneous networks. In *WSDM*. 741–749.
[53] S. Yan, D. Xu, B. Zhang, and H. Zhang. 2005. Graph embedding: A general framework for dimensionality reduction. In *CVPR*, Vol. 2. IEEE, 830–837.
[54] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang. 2015. Network representation learning with rich text information. In *IJCAI*.
[55] R. Ying, R. He, K. Chen, P. Eksombatchai, W. Hamilton, and J. Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*. 974–983.
[56] Ji. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, and J. Leskovec. 2019. Hierarchical temporal convolutional networks for dynamic recommender systems. In *WWW*. 2236–2246.
[57] X. Yu, X. Ren, Q. Gu, Y. Sun, and J. Han. 2013. Collaborative filtering with entity similarity regularization in heterogeneous information networks. *IJCAI HINA* 27 (2013).
[58] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM*. 283–292.
[59] X. Yu, X. Ren, Y. Sun, B. Sturt, U. Khandelwal, Q. Gu, B. Norick, and J. Han. 2013. Recommendation in heterogeneous information networks with implicit user feedback. In *RecSys*. 347–350.
[60] D. Zhang, J. Yin, X. Zhu, and C. Zhang. 2016. Homophily, structure, and content augmented network representation learning. In *ICDM*. IEEE, 609–618.
[61] X. Zhao, R. Louca, D. Hu, and L. Hong. 2018. Learning item-interaction embeddings for user recommendations. *arXiv preprint arXiv:1812.04407* (2018).
[62] D. Zheng, X. Song, C. Ma, Z. Tan, Z. Ye, J. Dong, H. Xiong, Z. Zhang, and G. Karypis. 2020. DGL-KE: Training Knowledge Graph Embeddings at Scale. In *SIGIR*. 739–748.
[63] Z. Zhu, S. Xu, J. Tang, and M. Qu. 2019. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *WWW*. 2494–2504.