KPAR: Knowledge-aware Path-based Attentive Recommender with Interpretability

LEIGH EYTAN, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel
VERONIKA BOGINA, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel
IRAD BEN-GAL, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel
NOAM KOENIGSTEIN, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel

Knowledge Graph (KG)-based recommender systems utilize both Collaborative Filtering (CF) data and informative KG data to improve prediction accuracy. Path-based KG methods are a family of KG-based recommendation models that explore the interlinks within a knowledge graph in order to enhance the connectivity between users and items with rich complementary information. A key advantage of path-based KG methods stems from their ability to enable intuitive explanations naturally. In this work, we present a novel path-based algorithm that employs neural attention in order to better extract the relevant information from the unified graph. Evaluations based on public KG recommendation datasets indicate a clear advantage to the proposed method compared to state-of-the-art path-based alternatives. Furthermore, we show that this advantage also extends to cold items where a better utilization of the KG leads to improved predictions in cases where no CF data is available. Finally, by performing attention-score analysis, we demonstrate the ability of our approach to provide better interpretability into the model's inner workings as well as extract more intuitive explanations. The code for this work is publicly available on GitHub: https://github.com/DeltaLabTLV/KPAR.

CCS Concepts: • **Computing methodologies** → **Causal reasoning and diagnostics**; **Machine learning**; • **Information systems** → **Recommender systems;**

Additional Key Words and Phrases: Knowledge graphs, neural networks, recommender systems, attention

ACM Reference Format:
Leigh Eytan, Veronika Bogina, Irad Ben-Gal, and Noam Koenigstein. 2025. KPAR: Knowledge-aware Path-based Attentive Recommender with Interpretability. *ACM Trans. Recomm. Syst.* 3, 3, Article 35 (March 2025), 23 pages. https://doi.org/10.1145/3673243

1 Introduction

Knowledge Graphs (**KGs**) provide a practical approach to representing large-scale information in multiple domains. A KG is a heterogeneous network consisting of multiple types of entities (nodes) and relations (edges). The graph's nodes represent different types of entities, and the graph's edges represent different types of relations between the entities. KGs have strong

This research was supported by the Israel Science Foundation grant 2243/20.
Authors' Contact Information: Leigh Eytan, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel; e-mail: leigh.eytan@gmail.com; Veronika Bogina, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel; e-mail: sveron@gmail.com; Irad Ben-Gal, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel; e-mail: bengal@tauex.tau.ac.il; Noam Koenigstein, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel; e-mail: noamk@tauex.tau.ac.il.

ACM 2770-6699/2025/03-ART35
https://doi.org/10.1145/3673243

representation abilities since multiple attributes of an entity can be obtained by following different edges on the graph.

Recommender systems employing KGs have attracted a considerable amount of research [13]. For example, a movies KG often consists of entities (nodes) such as movies, directors, actors, and so on., where the edges on the graph depict different types of relations e.g., *Actor X* $\xrightarrow{played\ in}$ *Movie Y*. Hence, KG-based recommender models can be seen as *hybrid* recommender models that utilize both **Collaborative Filtering** (**CF**) transactional data together with the KG meta-data in order to improve recommendations accuracy.

KG-based recommender models can be broadly categorized into three classes [13]: (1) *embedding-based methods*, (2) *path-based methods*, and (3) *propagation methods*. *Embedding-based methods* treat the KG graph as an extension of the bipartite graph induced by the CF transactional data. Vector embeddings are used to represent each node on the enhanced graph and direct connections are used as training data in order to learn these embeddings. In contrast, *path-based methods* employ multi-hop paths between a user and a target item in order to determine the user-item affinity scores. A key advantage of path-based methods stems from their inherent ability to explain recommendations [46]. Finally, *propagation methods* are based on the idea of embedding propagation. These methods refine the entity representation with the guidance of the connective structure in the KG.

In this article, we present **KPAR** - **K**nowledge-aware **P**ath-based **A**ttentive **R**ecommender- a novel path-based KG recommendations model. Different from state-of-the-art path-based methods that mostly make use of **Recurrent Neural Networks** (**RNNs**) [36, 46], KPAR employs neural attention in order to represent paths on the KG and rank them according to their relevancy. Neural self-attention techniques have recently gained much popularity after showing tremendous improvements over RNNs in multiple **Natural Language Processing** (**NLP**) tasks [8, 40]. Similarly, we show that the same trend can be repeated when considering paths for path-based KG recommendation models: by employing self-attention over the path's nodes and edges, KPAR achieves superior path representations which in turn leads to better accuracy.

Previous path-based models had no method of scoring the importance of each path and relied on simple aggregations of the paths in order to make the final prediction [36, 46]. Fundamentally, these models lacked a mechanism for determining the significance of each path with respect to the underlying prediction task. This led to an indiscriminate aggregation of all paths through mean pooling. In contrast, KPAR introduces a novel cross-attention mechanism for path representations that determines the degree of "attention" each path receives in the context of a specific user-item prediction task. Additionally, by highlighting the importance of each path to the ultimate prediction, this enhancement paves the way for improved interpretability and explanations.

Using publicly available KG recommendation datasets, we show that KPAR achieves superior results when compared to state-of-the-art path-based KG recommendation algorithms. Additionally, by leveraging KPAR's attention scores for the various pathways linking the user to the item, the model gains interpretability making the model's decision process more transparent. Interpretability in recommender systems has been linked to improved user trust, increased adoption, and fairness, as well as helping developers identify and correct mistakes [26, 38]. Moreover, we show that the main advantage of KPAR stems from better utilization of the KG information consequently leading to superior results on cold items. Finally, we show that KPAR can be generalized to support related tasks such as item similarity.

The rest of this article is organized as follows: In Section 2 we review related work. Section 3 gives an overview of our proposed model. The empirical results and additional experiments are described in Section 4, followed by a conclusion in Section 5.

2 Related Work

KGs have been broadly employed in the industry and academia for a long time [12, 37]. Much interest in KGs has been raised in 2012 following Google's presentation of a KG that semantically enhances their search engine through query expansion [33]. Consequently, KGs attracted an increasing amount of interest in the recommender systems community as well [13, 34]. In this context, KGs enable the embedding of auxiliary information such as **Content Based** (**CB**) data together with the usage of CF data. This forms a heterogeneous network consisting of multiple types of entities (nodes) and relations (edges) [27] where nodes represent entities such as *movies*, *users*, *actors*, *directors*, and so on., while the edges represent types of relations between the entities e.g., "*watched by*", "*acted in*", and "*directed by*", and so on. Hence, KG recommendation models can be seen as a sub-category of *hybrid models* which utilize both CF data together with CB data in order to improve recommendations accuracy [46].

KG recommender systems are broadly categorized into three groups: (1) *Embedding methods*, which use only the direct connections i.e., direct neighbors during training. (2) *Path-based methods*, which mine multiple paths i.e., indirect multi-hop connections, between users and target items in order to rank the items for recommendations. (3) *Propagation methods* that refine the entity representation on node's multi-hop neighbors. The KPAR model in this article belongs to the second category.

2.1 Embedding-based Methods

Embedding-based methods use the information of the KG directly to enrich and leverage the semantic representation of items and users. The information from the KG enables the enrichment of the user and item representations (embeddings) which are ultimately used in the recommendation process similarly to **matrix factorization models** (**MF**) [23]. In this way, embedding methods can be seen as an extension of MF that encodes the KG entities into low-rank embeddings together with the CF data [13]. For example, [53] proposed an embedding model that exploits items' structural knowledge and content knowledge (textual and visual) from a KG to learn better item latent representations. Later, [2] improved upon [53] by learning the embeddings of entities and relations in the graph using a novel distance metric between the graph's entities. Experiments show that both [53] and [2] are able to incorporate structural knowledge in order to boost performance and improve upon pure collaborative filtering.

2.2 Path-based Methods

One limitation of embedding-based approaches is their reliance on direct relations between entities, potentially overlooking valuable information from multi-hop relations. Additionally, embedding methods often lack the reasoning ability inherent to path-based techniques.

In contrast, path-based methods harness indirect connections within the knowledge graph by traversing multiple paths between users and target items. Early approaches like [25, 35, 52] employed predefined *meta-paths* that specified the format and length of paths in the KG. A meta-path represents a sequence of relations between object types, defining a composite relation between the starting and ending entities. For example, for entities authors (A), articles (P), and venues (V), a predefined meta-path could be "$VPAPV$", indicating venue-paper-author-paper-venue. However, these methods heavily depend on handcrafted meta-paths, necessitating manual tuning and domain expertise.

Moving beyond meta-paths, [36] introduced **Recurrent Knowledge Graph Embedding** (**RKGE**), employing RNNs for automatic path mining between users and items, eliminating the need for manual meta-path definitions. Similarly, [46] proposed **Knowledge-aware Path Recurrent Network** (**KPRN**), constructing path sequences using entity and relation embeddings.

KPRN utilizes **Long-Short Term Memory** (**LSTM**) units [19] to extract path representations, which are then used to compute user-item affinity scores.

Both RKGE and KPRN are considered state-of-the-art in path-based KG recommendation models. The KPAR model presented in this work follows the design principles of [36] and [46]. All these models comprise the following steps: path extraction, path embedding, path aggregation, and prediction.

The path extraction process in RKGE, KPRN, and KPAR is relatively similar. Specifically, KPAR , following [46], represents a path as a sequence of entity embeddings with interspersed relation type embeddings. For the path embedding step, RKGE uses a classical RNN and KPRN employs an LSTM network. In contrast, KPAR takes a different route by introducing a self-attention mechanism for path embedding, aligning with recent trends in neural models [11, 14, 20, 40].

In RKGE, path aggregation is achieved via a max-pooling operation, and the final prediction involves linear classification over the aggregated path representations. KPRN takes a different approach, computing separate predictive scores for each path using a two-layered fully-connected network. Subsequently, the prediction scores from all paths are aggregated through logarithmic-average pooling.

KPAR departs from these models by introducing a cross-attention mechanism that dynamically assigns attention weights to each path based on its relevance. Importantly, unlike the different pooling operations performed by previous models, cross-attention is a parameterized operation that *learns* to select the best paths according to the specific user-item prediction task at hand. Additionally, the attention weights offer insights into the significance of each path and enhance the model's interpretability and explanatory capabilities. We consider this novel cross-attention mechanism for path aggregation to be a significant advancement in path-based architectures. As we demonstrate later, this approach not only enhances performance but also facilitates better model interpretability and explanations.

2.3 Propagation Methods

Propagation methods are based on the idea of *embedding propagation* which enables entity representations to be guided by the connective structure of the KG [13]. RippleNet [41] extends the user's interests iteratively along KG to reveal additional potential interests of the user. **Knowledge Graph Attention** (**KGAT**) [44] recursively propagates the embeddings from a node's neighbors and employs neural attention to discriminate the importance of the neighbors.

Inspired by **Graph Convolution Network** (**GCN**) [22], **Knowledge Graph Convolutional Networks** (**KGCN**) [43], and **Knowledge-aware Graph Neural Networks with Label Smoothness regularization** (**KGNN-LS**) [42] learn item representations from distant neighbors inwardly (in contrast to RippleNet which works their way outwards). Both leverage the idea of GCN by sampling a fixed number of neighbors to create a *receptive field*, which makes the learning process highly efficient and scalable. These works show that the propogation approach of utilizing neighbors' information is capable of leading to a big boost in the recommendation task.

Collaborative Knowledge-aware Attentive Network (**CKAN**) [47] introduced a heterogeneous propagation strategy that encodes collaborative signals alongside knowledge associations. In addition, CKAN employs neural attention to discriminate the contribution of different knowledge-based neighbors. Finally, **Knowledge Graph Intent Network** (**KGIN**) [45] improves upon KGAT by considering the intentions behind a user-item interaction, using an auxiliary item knowledge. KGIN intorduced a new information aggregation scheme for GNNs, which recursively integrates the relation sequences of long-range paths. This approach allows to distill useful information about user intents and encode them into the representations of users and items. KGIN was shown to outperform many state-of-the-art KG models including recent propagation models such as KGAT [44], KGNN-LS [42], and CKAN [48].

2.4 Self-supervised Learning Techniques

In recent research on KG-based recommender systems, there has been a growing interest in the development of innovative self-supervised learning techniques aimed at enhancing entity representations within KGs. These techniques offer promising solutions to challenges related to sparsity, noise, and information integration.

Multi-level Cross-view Contrastive Learning for Knowledge-aware Recommender System (**MCCLK**) [55] introduces graph contrastive learning for KGs to reduce potential knowledge noise. KG contrastive signals are further used to guide the user preference learning.

Knowledge Graph Contrastive Learning (**KGCL**) [50] is the Knowledge Graph Contrastive Learning framework designed to address issues related to the sparsity of long-tail entities and noise within KGs. It leverages a topological denoising framework to introduce additional supervision signals that guide its cross-view contrastive learning paradigm.

In Reference [6], a graph contrastive learning approach is introduced to treat information from descriptive attributes and structural connections as two distinct modalities. Focusing on the task of scholarly recommendations, the model aims at learning informative node representations by maximizing the agreement between the descriptive view and the structural view.

Self-Supervised Attentive **Knowledge Rationalization** (**KGRec**) [49] introduces a self-supervised attentive knowledge rationalization mechanism that integrates generative and contrastive self-supervised tasks for recommendation through rational masking. By prioritizing important knowledge with high rational scores, KGRec is trained to reconstruct and emphasize valuable knowledge connections. Additionally, a contrastive learning task aligns signals from both the knowledge and user-item interaction views.

These self-supervised techniques have not yet been explored in the context of path-based KG models, such as the one presented in this article. Future research may explore the incorporation of ideas from the aforementioned articles to enhance the performance of our model. However, such investigations are left for future research.

KGCL [44] introduces graph contrastive learning for KGs to reduce potential knowledge noise. KG contrastive signals are further used to guide the user preference learning.

2.5 Explainable Recommendations

Many recommender systems operate as black boxes which provide no transparency into the inner workings of the recommendation process [3, 39]. However, explanations of recommendations can provide transparency by revealing the reasoning behind a recommendation and clarifying the validity of recommendations [1]. In addition, it has been shown that providing explanations along with recommendations helps improve user trust [16, 32].

Explainability over a KG is one of its natural benefits derived from the structure of the graph which enables more intuitive explanations. In particular, path-based methods are useful in providing explanations, since they exploit the graph's structure and display the connections between the entities over the graph. Explaining path-based KG recommender systems is achieved by following the paths that connect the user to the item [36, 46]. KPAR improves this process by employing a novel attentive path aggregation that determines the "importance" each path with respect to the user-item prediction task.

3 Proposed Model

Next, we present the KPAR model in detail. First, in Section 3.1 we provide notations and preliminaries. Then, in Section 3.2, we describe the model's architecture and its objective. Finally, in Section 3.3, we discuss specific implementation choices.

3.1 Preliminaries

3.1.1 The Knowledge Graph. A KG is a directed graph whose nodes are entities $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ connected by different types of directed edges. We denote by $\mathcal{R} = \{r_1, r_2, \ldots, r_g\}$ the set of types of relations between nodes. Formally, we define a KG as $\mathcal{KG} = \{(h, r, t) \mid h, t \in \mathcal{E}, r \in \mathcal{R}\}$, where each triplet (h, r, t) indicates the existence of a relationship of type r from the *head* entity h to *tail* entity t. In recommender systems, $\mathcal{KG}$ usually consists of items meta-data and sometimes user data such as demographics. Specifically, the knowledge graphs employed in this research consist of nodes that represent *persons* that contribute to the creation of the items. For example, in the movies domain, the KG consists of *persons* such as *actors*, *directors*, *producers*, and so on., and the relation types are *"acted in"*, *"directed by"*, *"produced by"*, accordingly. Similarly, in the music domain, the KG consists of the following types of *persons*: *artists*, *composer*, *lyricist*, and the relations are *"sung by"*, *"composed by"*, *"written by"*, accordingly. Note that the KG is a directional graph, hence two directional relations exist between an item and a person i.e., *"sung by"* and *"sang"* for the relation from the song to the singer and the vice versa.

3.1.2 The Collaborative Filtering Graph. We denote by $\mathcal{U} = \{u_k\}_{k=t}^{M}$ and $\mathcal{I} = \{i_t\}_{t=1}^{N}$ the sets of *users* and *items*, where M and N are the number of users and the number of items, respectively. In order to represent the interaction between user u and item i in the CF graph, we use the triplet: $(u, interacted\ with, i) \in \mathcal{CF}$. Here, the directed relation between the user u and the item i is of type *"interacted with"* which may mean a click, a purchase, watching a movie, and so on. Similarly, the corresponding relation, namely that an item i has been played, watched, purchased, and so on. by a user u is denoted by $(i, was\ interacted\ by, u) \in \mathcal{CF}$.

3.1.3 The Unified Graph. We merge the KG graph $\mathcal{KG}$ with the CF graph $\mathcal{CF}$ in order to form a unified graph $\mathcal{G}$ as follows: $\mathcal{G} = \mathcal{KG} \cup \mathcal{CF}$. Namely, the unified graph $\mathcal{G}$ includes all the directed triplets from both the knowledge graph $\mathcal{KG}$ as well as the collaborative filtering data in $\mathcal{CF}$. Hence, the unified graph's entities (nodes) are the *users* and *items* from $\mathcal{CF}$ and the *persons* from $\mathcal{KG}$, which are all connected by different types of relations (edges) from both graphs.

In the unified graph, connections between entities are given by directed paths which are finite sequences of nodes and edges leading from the *source* entity to the *target* entity. A *direct* connection is a connection of two entities via a path of length 1. For example: *Uma Thurman* $\xrightarrow{acted\ in}$ *Pulp Fiction* or *user x* $\xrightarrow{interacted\ with}$ *item y*. In other words, a direct connection between entities is based on the existence of a directed edge (a triplet) between these two entities which implies the existence of a fact that connects the items.

An *indirect connection* between entities is formed by a *path* of length > 1 in the unified graph $\mathcal{G}$. For example: *Uma Thurman* $\xrightarrow{acted\ in}$ *Pulp Fiction* $\xrightarrow{was\ directed\ by}$ *Quentin Tarantino*. We denote by $\mathcal{P}(e_i, e_j) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\}$ a set of k different paths from entity e_i to entity e_j, where $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k$ are the paths.

3.1.4 Problem Formulation. The general objective of a path-based KG recommender model is as follows: Given a user u, a target item i, and a set of paths that connect them $\mathcal{P}(u, i) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\}$, the goal of a path-based model is to predict the affinity of user u to the item i based on the set of paths $\mathcal{P}(u, i)$ according to

$$\hat{y}_{ui} = f_\theta(u, i \mid \mathcal{P}(u, i)), \tag{1}$$

where f_θ denotes the recommender model parameterized by θ, and $\hat{y}_{ui}$ is the predicted affinity score for the user-item interaction.

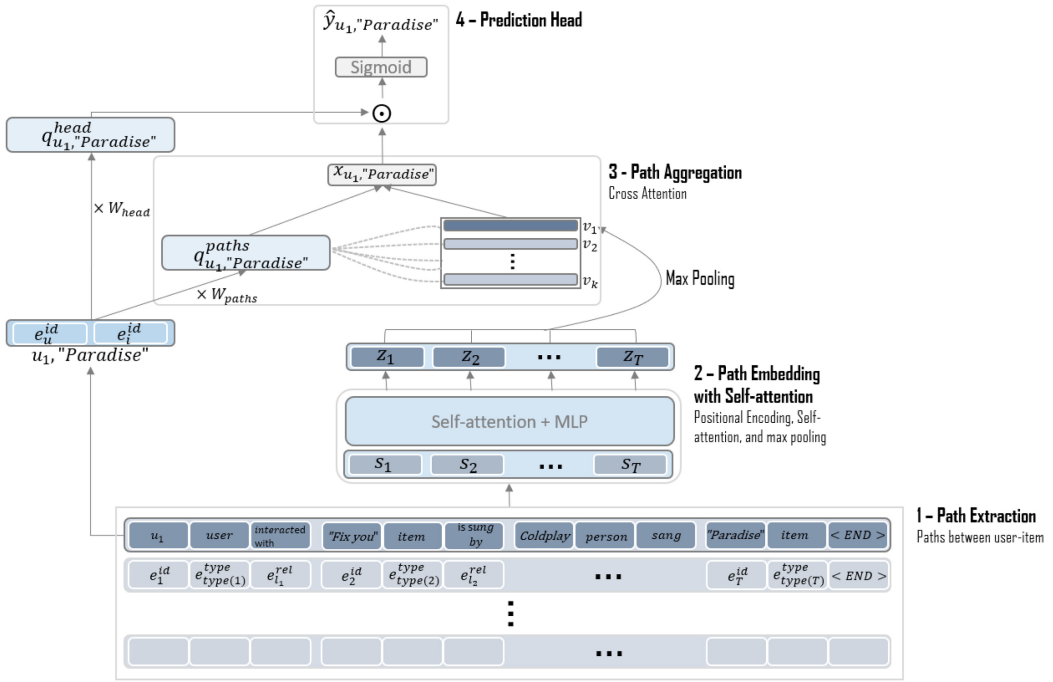


Fig. 1. A schematic illustration of KPAR . The model is comprised of four components: (1) path extraction and tokenization, (2) a path-embedding layer based on self-attention, (3) path aggregation via cross-attention, and (4) a prediction head to infer the ultimate user-item affinity score.

3.2 The KPAR Model

The KPAR model is comprised of four components: (1) path extraction and tokenization, (2) path embedding via self-attention, (3) path aggregation via cross-attention, and finally (4) a prediction head which infers the user-item affinity score. In what follows, we shall describe each component in detail and explain the reasoning for its architectural choices. A schematic illustration of KPAR is shown in Figure 1.

3.2.1 Path Extraction. The first step in any path-based KG algorithm is to extract multiple paths that connect the source and the target entities [36, 46]. The set of connecting paths serves as the input for the model. We consider the set of paths $\mathcal{P}(u, i)$ which connect user u with item i. Since the number of paths between a user and an item can be exponentially large, path-based methods rely on sampling paths according to some criteria. Specifically, it has been shown that shorter paths suffice to model entity relations, whereas longer paths tend to lose semantic meanings and introduce noise [35]. Hence, a common practice is to define a threshold limit T on the paths' length.

Once the set of relevant paths has been extracted, KPAR performs path tokenization. Each path is broken into entity-relation tuples as follows: The first token is a tuple consisting of the source entity (the user) followed by the first relation. Similarly, the second token consists of the second entity followed by the second relation, and so on. The last token consists of the target entity (the target item) followed by the special $< END >$ symbol representing the end of the path.

In KPAR, each token is an embedding that consists of a concatenation of three smaller embeddings: (1) an entity (node) id representation $\boldsymbol{e}_i^{id}$ which encodes each entity in the graph. Formally,

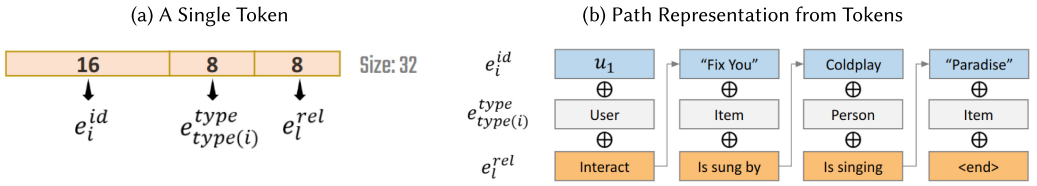


Fig. 2. (a) A single token in KPAR is a concatenation of three smaller representations: (1) e_i^{id}, the node embedding, (2) $\boldsymbol{e}^{type}_{type(i)}$, the node's type embedding, and (3) e_l^{rel}, the relation type embedding. (b) An illustration of a path consisting of 4 tokens: $u_1 \xrightarrow{interacted\ with}$ *"Fix You"* $\xrightarrow{is\ sung\ by}$ *Coldplay* $\xrightarrow{sang}$ *"Paradise"*. Note that in the last token, the e_l^{rel} representation is occupied by the special $< END >$ symbol signalling the end of the path.

$\boldsymbol{e}_i^{id} \in \mathbb{R}^{d_{id}}$ is a d_{id}-dimensional vector that represents entity i. The $\boldsymbol{e}_i^{id}$ vector is followed by a concatenation of (2) an entity type embedding $\boldsymbol{e}^{type}_{type(i)}$ which encodes the entity type i.e., *user*, *item*, *person* of entity i. Formally, $\boldsymbol{e}^{type}_{type(i)} \in \mathbb{R}^{d_{type}}$ is a d_{type}-dimensional vector representing the entity type, where $type(i)$ denotes the entity type of entity i. Finally, the last embedding in the token concatenation is (3) a relation type embedding $\boldsymbol{e}_l^{rel}$ which encodes the type of relation that leads to the next entity. Formally, $\boldsymbol{e}_l^{rel} \in \mathbb{R}^{d_{rel}}$ is the relation embedding for a relation type l. Remember that the path ends with the entity of the target item followed by the special $< END >$ representation. The $< END >$ representation is a special d_{rel}-dimensional representation that comes in place of the last (missing) relation type in the last token in order to denote the end of the sequence. To summarize, the i'th token, denoted by $\boldsymbol{t}_i \in \mathbb{R}^d$, is a d-dimensional embedding which is a concatenated representation consisting of three different smaller embeddings as follows:

$$\boldsymbol{t}_i = \boldsymbol{e}_i^{id} \oplus \boldsymbol{e}^{type}_{type(i)} \oplus \boldsymbol{e}_l^{rel}, \tag{2}$$

where $d = d_{id} + d_{type} + d_{rel}$ and $\oplus$ denotes concatenation.

In our implementation, $d_{id} = 16$, $d_{type} = 8$, and $d_{rel} = 8$. Hence, $d = 32$ is the dimensionality of each token. Figure 2(a) depicts a token representation in KPAR , while Figure 2(b) depicts a single path representation consisting of 4 tokens. Finally, we note that the motivation to include relation type representations stems from [46], which showed that encoding relation types improve overall path representations in path-based models.

3.2.2 Path Embedding. KPAR learns path embeddings by utilizing self-attention across the path's tokens, similar to the method a Transformer uses to derive sentence representations from individual words [10, 40]. This is different from previous path-based recommendation models [36, 46] which relied on RNNs to represent paths. Self-attention is an alternative to RNNs which yields superior results while reducing training time by enabling better parallelization [20, 40]. However, since tokens are being processed in parallel, positional encoding needs to be actively injected into the tokens in order to utilize positional information.

As explained above, a path of length T consists of entity-relation tuples represented by token embeddings: $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_T$ (according to) Equation (2)). KPAR employs positional encoding using the sine and cosine functions. For a given token $\boldsymbol{t}_i \in \mathbb{R}^d$, a positional encoding vector $\boldsymbol{y}_i \in \mathbb{R}^d$ is computed and added to $\boldsymbol{t}_i$. Specifically, at position j, the scalar $y_i[j]$ from the positional vector $\boldsymbol{y}_i$

is given by

$$y_i[j] = \begin{cases} \sin\left(\frac{i}{10,000^{\frac{2j}{d}}}\right), & \text{if } j \text{ is even,} \\ \cos\left(\frac{i}{10,000^{\frac{2j}{d}}}\right), & \text{otherwise} \end{cases} \tag{3}$$

For the $i'th$ token $\boldsymbol{t}_i$, we denote the position-aware representation by $\boldsymbol{s}_i \in \mathbb{R}^d$ which is given by: $\boldsymbol{s}_i = \boldsymbol{t}_i + \boldsymbol{y}_i$.

The position-aware tokens $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_T$ are fed into a multi-head self-attention unit in order to achieve "contextualized" tokens: $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T$. Formally, we stack the position-aware tokens into a matrix $\boldsymbol{S} \in \mathbb{R}^{T \times d}$ and employ multi-head self-attention: $Y = MultiHead(\boldsymbol{S})$, which yields $Y \in \mathbb{R}^{T \times d}$ consisting of the T "contextualized" tokens $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T$. The multi-head attention operation follows that of [40].

Next, each "contextualized" token is passed through a feed-forward **multi-layer perceptron** (**MLP**) which is applied on each token separately and identically as in [10, 40]. Our MLP consists of two fully-connected layers with a ReLU activation in between:

$$\boldsymbol{z}_i = \max\{0, \boldsymbol{y}_i \boldsymbol{W}_1 + \boldsymbol{b}_1\}\boldsymbol{W}_2 + \boldsymbol{b}_2, \tag{4}$$

where where $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d \times d}$ are learned transformation matrices, $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{R}^d$ are the learned biases, and $\boldsymbol{z}_i \in \mathbb{R}^d$ is the output. Finally, the tokens $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T$ are reduced to a d dimensional path representation $\boldsymbol{v} \in \mathbb{R}^d$ using mean-pooling: $\boldsymbol{v} = max\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T\}$. Hence, the output of the path-embedding component is a single d-dimensional path representation $\boldsymbol{v} \in \mathbb{R}^d$ for each path.

3.2.3 Path Aggregation. Recapping so far, KPAR starts with a set of sampled user-item paths $\mathcal{P}(u, i) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\}$. Every single path $\boldsymbol{p}_i$ is represented by a sequence of tokens $\{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_T\}$. The path representation component of KPAR takes each such path $\boldsymbol{p}_i$ and yields a vectorized path representation $\boldsymbol{v}_i \in \mathbb{R}^d$. Hence, at this point, we are left with a set of vectorized path representations $\mathcal{V}(u, i) = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}^\intercal$ corresponding to the original paths in $\mathcal{P}(u, i)$. Next, KPAR employs cross-attention in order to aggregate the path representations based on their relative importance to the prediction task.

Remember that our ultimate goal is to score the affinity between a user u and an item i based on the set of sampled paths. Naturally, some paths may be more informative than others with respect to this objective. KPAR learns to assign importance weights to each path with respect to the user-item prediction task at hand. To this end, the user-item embeddings are used to create a query vector $\boldsymbol{q}_{ui}^{paths}$ over the paths as follows: Given a user u and an item i, the query $\boldsymbol{q}_{ui}^{paths}$ is given by $\boldsymbol{q}_{ui}^{paths} = \boldsymbol{W}_{paths}(\boldsymbol{e}_u^{id} \oplus \boldsymbol{e}_i^{id})$, where $\boldsymbol{W}_{paths} \in \mathbb{R}^{d \times 2 \cdot d_{id}}$ is a learnable projection matrix that projects the concatenation of the user embedding $\boldsymbol{e}_u^{id}$ and the item embedding $\boldsymbol{e}_i^{id}$ into a d-dimensional cross-attention space.

In order to aggregate the paths according to their relevance, the cross-attention is performed between $\mathcal{V}(u, i) = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}^\intercal$, the set of k path representations, and the query $\boldsymbol{q}_{ui}^{paths}$ as follows:

$$\boldsymbol{x}_{ui} = Attention(\boldsymbol{q}_{ui}^{paths}, \mathcal{V}(u, i)\boldsymbol{W}_{ck}, \mathcal{V}(u, i)\boldsymbol{W}_{cv}), \tag{5}$$

where $\boldsymbol{W}_{ck} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{W}_{cv} \in \mathbb{R}^{d \times d}$ are learnable projection matrices, and the attention operation follows that of [40].

The output $\boldsymbol{x}_{ui} \in \mathbb{R}^d$ (Equation (5)) encodes the relevant information from all the paths in $\mathcal{P}(u, i)$ according to their relative importance with respect to the query. Note that from the

cross-attention operation above, we can also extract attention scores that quantify how much "attention" the model gives to each of the paths. Later, in Section 4.7, we show how these attention scores can be harnessed to generate clear and intuitive explanations for the model's predictions.

3.2.4 The Prediction Head. KPAR treats the recommendation learning task as a binary classification problem where a set of paths between a positive user-item interaction are assigned the label 1 and paths between a "negative" user-item interaction (no interaction) are marked by the label 0. KPAR's loss function is based on the **Binary Cross Entropy** (**BCE**) loss, with L2 regularization on the trainable parameters, collectively denoted by $\boldsymbol{\Theta}$. Formally, given a training dataset D, where each datum $(u, i, y_{ui}) \in D$ consist of a user u, an item i, and a label $y_{ui} \in \{0, 1\}$, KPAR's training objective is given by

$$\mathcal{L} = -\frac{1}{|D|} \sum_{(u,i,y_{ui}) \in D} \left[y_{ui} \cdot \log(\hat{y}_{ui}) + (1 - y_{ui}) \cdot \log(1 - \hat{y}_{ui}) \right] + \lambda \left\| \boldsymbol{\Theta} \right\|^2, \tag{6}$$

where $\hat{y}_{ui}$ are KPAR's predictions for a user u and an item i, $|D|$ is the number of data points in D, and λ is a hyperparameter.

In order to generate its predictions, KPAR's prediction head employs a dot-product between the paths' aggregation $\boldsymbol{x}_{ui}$ and a prediction-head query $\boldsymbol{q}_{ui}^{head}$, which is given by a projection of the user and item representations on a learnable matrix. Formally, $\boldsymbol{q}_{ui}^{head}$, the prediction-head query for a user u and and item i, is given by $\boldsymbol{q}_{u,i}^{head} = \boldsymbol{W}_{head}(\boldsymbol{e}_u \oplus \boldsymbol{e}_i)$, where $\boldsymbol{W}_{head} \in \mathbb{R}^{d \times 2 \cdot d_{id}}$ is a learnable matrix. Finally, the model's prediction is given by: $\hat{y}_{ui} = \sigma(\boldsymbol{x}_{ui}^{\intercal} \boldsymbol{q}_{ui}^{head})$, where $\sigma(a) \stackrel{\Delta}{=} \frac{1}{1+e^{-a}}$ is the sigmoid function.

We summarize the key notations used in this article in Table 1.

3.3 Implementation Details

For the sake of reproducibility, we wish to discuss some implementation details used in this article. The KPAR instance used in this article is based on the following settings:

(1) **KPAR Architectural Settings:** As mentioned earlier, we implemented a classic transformer that resembles the encoder from [40]. In our implementation, the transformer stack consisted of a single layer with 8 attention heads. The embeddings dimensionality was set to $d_{id} = 16$, $d_{type} = 8$, and $d_{rel} = 8$. Hence, $d = d_{id} + d_{type} + d_{rel} = 32$ was the dimensionality along the rest of the model. Finally, to reduce complexity we used $\boldsymbol{W} = \boldsymbol{W}_{paths} = \boldsymbol{W}_{head}$, and $\boldsymbol{W}_{ck} = \boldsymbol{W}_{cv} = \boldsymbol{I}_{d \times d}$ were fixed to the d-dimensional identity matrix ($\boldsymbol{I}_{d \times d}$).

(2) **Path Extraction:** Path sampling in KPAR can follow any sampling scheme. In this article, we followed the scheme described in [46] which samples paths uniformly. Additionally, we set the limit of path length to $T = 4$, i.e., each path includes 4 entities and 3 relations. Negative paths were sampled in order to create negative examples for the training data. The negative paths are paths that start at the user and traverse randomly (uniform random walk) to an item that the user did not interact with i.e., there is no direct edge between the user and the item. For each positive example, we dynamically sampled 4 negative examples which were resampled on every training epoch.

(3) **Optimization Details:** Optimization is achieved by following stochastic gradient descent using an Adam optimizer [21] with a batch size of 128, with the learning rate of 0.0001 and L2 regularization coefficient of 0.001.

(4) **Hyperparameters:** To determine the optimal hyperparameters for our model, we employed the Optuna[1] library and conducted tree-structured Parzen optimization [5]. The following

[1] https://optuna.readthedocs.io/en/stable/

Table 1. Summary of Key Notations

Notation	Definition	Dimension
$\mathcal{KG}$	Knowledge graph	
$\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$	Entities in the KG	k
$\mathcal{R} = \{r_1, r_2, \ldots, r_g\}$	Relation between two entities	g
$\mathcal{U} = \{u_k\}_{k=t}^{M}$	A set of users	M
$\mathcal{I} = \{i_t\}_{t=1}^{N}$	A set of items	N
$\mathcal{CF}$	Collaborative Filtering graph	$M \times N$
$\mathcal{G} = \mathcal{KG} \cup \mathcal{CF}$	A Unified graph	
$\mathcal{P}(u,i) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\}$	A set of paths that connect between a user u and an item i	k
f_θ	A model parameterized by θ	
$\hat{y}_{ui}$	The predicted score for the user-item interaction	
D	A training dataset	
$\boldsymbol{e}_i^{id}$	A vector that represents entity i	d_{id}
$\boldsymbol{e}_{type(i)}^{type}$	An entity type embedding	d_{type}
$\boldsymbol{e}_l^{rel}$	A relation type embedding	d_{rel}
t_i	A token embedding	$d = d_{id} + d_{type} + d_{rel}$
y_i	A positional encoding vector	d
s_i	A path token	d
z_i	A contextualized token at position i	d
v_i	A vectorized path representation	d
x_{ui}	An aggregated representation of all path from u to i	d
A	A matrix with the input vectors	$n \times d$
W^K, W^V, W^Q	Learnable matrices in self-attention.	$d \times d_k$, $d \times d_v$, $d \times d_k$
W_{cv}	A learnable projection matrix in cross-attention.	$d \times d$
W_{ck}	A learnable projection matrix in cross-attention.	$d \times d$
W_{paths}	A learnable projection matrix that projects the concatenation of the user and the item embeddings into d-dimensional space of the path representations	$d \times 2 \cdot d_{id}$

parameters were fine-tuned: Embedding dimension was optimized within the range of $[10, 100]$, learning rate was explored across $[10^{-6}, 10^{-3}]$, and weight decay regularization was adjusted within $[10^{-5}, 10^{-1}]$.

Note that the code for KPAR is publicly available on our Git repository.

3.4 Computational Complexity

In what follows, we provide an analysis of the computational complexity of KPAR's components and its overall complexity:

(1) **Path Extraction:** The path-extraction process is highly dependant on the properties of the KG in use. In general finding paths between two nodes on a graph can be a challenging problem. This however is a common challenge for all path-based approaches where training and inference is based on sets of paths connecting a user to a target item. A brute force approach can be exponentially expensive, but setting a threshold on paths' length is an effective and sufficient mitigation. In this work, we followed [46] and employed a brute force approach with a threshold of $T = 4$ on paths' length. Limiting paths' length agrees with the earlier observation from Section 3.2.1 that shorter paths are generally more informative whereas

longer paths tend to lose semantic meanings and introduce noise. Assuming the outdegree of the node is δ, the complexity of this step is $O(\delta^T)$. Further improvements to this step may be achieved by employing a backtracking search approach [24].

(2) **Path Embedding:** The path-embedding of KPAR is based on a transformer unit. The complexity of which is dependant on the path length and d, the dimensionality of the internal representation. For a single attention head this means a complexity of $O(T^2d)$ for the self-attention operation and $O(Td^2)$ for the MLP. Hence, for h attention heads we get a total of $O(h(T^2d + Td^2))$.

It is interesting to compare the complexity of the path aggregation step in KPAR to the same step in RNN-based alternatives such as [36, 46]. The complexity of employing h RNN units in parallel on the path of length T consisting of d-dimensional tokens is $O(hTd^2)$. Note that this complexity is similar to the MLP component of KPAR . The additional cost for path-embedding in KPAR is the factor $O(T^2d)$ which stems from the self-attention operation. However, remembering that $T = 4$ and $d = 32$, we easily see that the dominant part in the path-embedding step of KPAR is comparable to that of RNN based models such as [36, 46].

(3) **Path Aggregation:** The path-aggregation complexity of KPAR is based on a single cross-attention operation between a user-item query and k paths. The query and paths are all represented by d-dimensional vectors. Hence, the complexity of this step is $O(kd^2)$. Previous path-based models employed simple pooling operations which take $O(kd)$. Hence, for the path-aggregation phase, KPAR carries an additional complexity factor of d compared to path-aggregation in previous models. However, remember that employing this cross-attention step over the paths, enables KPAR to quantify the relative importance of each path with respect to the user-item prediction task. In Section 4.4 we provide an ablation study which demonstrates the significant contribution of our cross-attention approach compared to the simple pooling approach of previous models [36, 46].

(4) **Prediction Head:** Finally, KPAR employs a simple linear classification on the d-dimensional representation of the aggregated paths with complexity of $O(d)$. Clearly, the complexity of the prediction head is relatively negligible compared to that of the rest of the model. In addition, the complexity of this step is similar to the complexity of the prediction step in alternative models [36, 46].

In summary, our model's complexity is comparable to that of previous path-based models relying on RNNs. KPAR introduces additional complexity in both the path-embedding and path-aggregation steps due to the incorporation of self-attention and cross-attention mechanisms, respectively. It's worth noting that in the path-embedding step, the added computational cost is a non-dominant additive term, which has negligible practical impact. The primary differentiation between KPAR and its RNN-based counterparts lies in the path-aggregation step, where an additional factor d is introduced. This factor arises from our use of cross-attention for path-aggregation instead of a simple pooling operation. However, as demonstrated later, the adoption of cross-attention not only significantly enhances model accuracy but also enables interpretability, facilitating the generation of user-friendly explanations.

4 Experiments

We evaluate the KPAR model using publicly available KG recommendations datasets. We begin with an investigation of the prediction accuracy with respect to state-of-the-art path-based baselines and an ablation study (Section 4.4). In addition, we explore the ability of KPAR to employ information from the KG in order to recommend completely cold items, i.e., items without any

Table 2. Dataset Statistics

Datasets		MI	KKBox	Last-FM
User-Item Interaction	# Users	6,040	10,000	23,566
	# Items	3,883	210,738	48,123
	# Interactions	1,000,209	2,382,140	3,034,796
Knowledge Graph	# Entities	20,641	308,694	58,266
	# Entity Types	3	3	3
	# Relation Types	6	6	9
	# Triplets	1,020,996	2,821,357	464,567

user interaction (Section 4.6). Then, we demonstrate the model's ability to produce simple and intuitive explanations (Section 4.7). Finally, we briefly demonstrate the model's ability to extract explainable item-to-item similarities (Section 4.8).

4.1 Datasets

We evaluate KPAR on two publicly available KG recommendation datasets:

- **MovieLens-1M and IMDB (MI):** The MovieLens-1M CF dataset [15] together with the IMDB[2] KG dataset in which the movies (items) are linked by their titles and release dates. The MovieLens-1M dataset provides user-item interactions, while IMDB provides the auxiliary KG information on movies such as actors, directors, and writers.
- **KKBox:** The KKBox dataset is taken from the WSDM Cup 2018 Challenge[3] [9]. KKBox is a music streaming service consisting of user-item interactions together with a KG of music meta-data which includes singers, composers, and lyricists.
- **Last-FM:** This is the music listening dataset collected from Last.fm[4] online music systems. Wherein, the tracks are viewed as the items. In particular, we take the subset of the dataset released in KGAT [44] and subsequently used by [45].

The dataset statistics are summarized in Table 2. Similar to prior research efforts [18, 36, 46, 51], we treat all user interactions (e.g., rating a movie or listening to a song) as positive examples. In both datasets, we randomly divided the data into training (80%) and test (20%) sets. Additionally, we set aside an extra 10% of the training data to construct a validation set. The validation set played a crucial role in hyperparameter tuning for KPAR and all baseline models. After optimizing the hyperparameters, we reintegrated the validation set into the training data and constructed the final models using the original 80% of the data. These final models were then evaluated on the remaining 20% of the data. Following [46], to create a balanced evaluation setup, we employed the "one-plus-random" approach [4] and sampled 100 negative interactions for each positive interaction in the test set.

4.2 Baseline and Ablation Models

We focus our evaluation on path-based models and a recent propagation model for completeness:

- **RKGE** [36]: RKGE is a state-of-the-art path-based KG model that employs a bidirectional RNN to extract multiple paths between users and items. It has demonstrated superior performance over well-known baselines, including BPR [31], NCF [18], LIBFM [30], HeteRS [28],

[2] https://www.imdb.com/
[3] https://www.kaggle.com/competitions/kkbox-music-recommendation-challenge/data
[4] https://grouplens.org/datasets/hetrec-2011/

HeteRec [52], GraphLF [7], and CKE [53]. The following hyperparameter settings were optimized: the latent dimension of factors across [10, 20, 50, 100, 200], while the learning rate and regularization coefficient across $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$.

— **KPRN** [46]: KPRN employs LSTM networks to process a set of sampled paths. It enhances path representation by incorporating relation type embeddings. KPRN outperformed several prominent baselines, including BPR [31], NFM [17], CKE [53], and FMG [54]. The following hyperparameter settings were optimized: the learning rate across [0.001, 0.002, 0.01, 0.02], regularization coefficient across $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$. Other parameters were set according to their values in [46].

— **KGIN** [45]: KGIN is a state-of-the-art propagation model designed to identify user-item relations with a fine-grained level of intent. It leverages relation dependencies to preserve the semantics of long-range connectivity. Notably, KGIN has shown superior performance compared to state-of-the-art propagation models including KGAT [44], KGNN-LS [42], and CKAN [47]. We adhered to the original article's approach [45] and maintained identical hyperparameters across all datasets, adopting the same values as outlined in [45], except for the learning rate, which underwent optimization within the range $[10^{-6}, 10^{-3}]$.

— **KPRN-Transformer**: This baseline model is a hybrid of KPAR and the KPRN model introduced in [46]. It replaces KPRN's LSTM component with a self-attention mechanism, similar to that of KPAR, while excluding KPAR's path aggregation component. This baseline serves as an ablated version of KPAR that helps assess the relative impact of KPAR's path-embedding via self-attention to its attentive path aggregation via cross-attention. Hyperparameters were optimized in a manner similar to that of KPAR .

The hyperparameters for both KPAR and the baseline models underwent optimization using the Optuna[5] library via tree-structured Parzen optimization [5] on the validation set, as explained above. Detailed information regarding the optimal hyperparameter configurations for all models is available in our Git repository.[6]

4.3 Evaluation Metrics

We report results in terms of the following metrics as defined in [32]:

— **Hit Rate at K (Hit@K)** - Measures if any relevant item was retrieved within the top-K positions of the recommendation list or not. Hit@K ignores the number of relevant items in the top-K, as long as there is at least one relevant item. As a consequence, Hit@K improves (increases) with K, the length of the recommendation list. We report an average across all test users.

— **Precision at K (Precision@K)** - Measures the proportion of the relevant items out of the top-K positions of the recommendation list. Different from Hit@K, Precision@K improves as more relevant items make it to the top-K recommendation list; however, Precision@K may decrease as the length of the recommendation list (K) increases. We report an average across all test users.

— **Recall at K (Recall@K)** - Measures the proportion of relevant items in the top-K positions in the recommendations list, out of the total relevant items of the user. We report an average across all test users.

— **Normalized Discounted Cumulative Gain at K (nDCG@K)** - The previous metrics do not consider the order in which the items are retrieved within the recommendation list.

[5] https://optuna.readthedocs.io/en/stable/

[6] These parameters are documented in the 'model_args.txt' file within the project's Git repository.

Table 3. Comparison of Different Methods on the hit@k Metric Across Movielens, KKBox, and LastFM Datasets

	MovieLens - Hit@k				KKBox - Hit@k				LastFM - Hit@k			
Method	@1	@5	@10	@20	@1	@5	@10	@20	@1	@5	@10	@20
RKGE	0.218	0.600	0.793	0.928	0.412	0.822	0.926	0.974	0.085	*0.488*	*0.800*	*0.875*
KPRN	0.266	0.658	0.830	0.941	0.512	0.858	0.935	0.975	0.081	0.288	0.422	0.566
KPRN-Trans.	0.279	0.668	0.822	0.931	*0.587*	*0.888*	*0.946*	*0.978*	*0.171*	0.386	0.516	0.643
KGIN	*0.395*	*0.781*	*0.885*	*0.949*	0.172	0.419	0.549	0.671	0.080	0.202	0.278	0.373
KPAR	**0.498**	**0.873**	**0.950**	**0.984**	**0.643**	**0.912**	**0.956**	**0.980**	**0.184**	**0.642**	**0.809**	**0.926**

The best result is highlighted in **bold** and the second best is highlighted in *italics*.

Table 4. Comparison of Different Methods on the nDCG@k Metric Across Movielens, KKBox, and LastFM Datasets

	MovieLens - nDCG@k				KKBox - nDCG@k				LastFM - nDCG@k			
Method	@1	@5	@10	@20	@1	@5	@10	@20	@1	@5	@10	@20
RKGE	0.298	0.262	0.256	0.267	0.512	0.485	0.478	0.479	0.085	**0.302**	*0.228*	0.176
KPRN	0.266	0.218	0.204	0.209	0.512	0.458	0.437	0.425	0.081	0.098	0.104	0.112
KPRN-Trans.	0.279	0.231	0.215	0.217	*0.587*	*0.548*	*0.533*	*0.519*	*0.171*	*0.159*	0.153	*0.148*
KGIN	*0.395*	*0.354*	*0.332*	*0.325*	0.172	0.159	0.156	0.155	0.081	0.074	0.077	0.084
KPAR	**0.498**	**0.408**	**0.373**	**0.362**	**0.643**	**0.598**	**0.577**	**0.559**	**0.184**	0.246	**0.265**	**0.287**

The best result is highlighted in **bold** and the second best is highlighted in *italics*.

In contrast, nDCG@K gives preference to solutions that place the relevant items at the top of the recommendation list. Normalization ensures that nDCG@K varies between 0 (low score) and 1 (the best score). We report an average across all test users.

— **Mean Percentile Rank (MPR)** - The previous metrics are user-centric [29] metrics that only the top-K recommendations. However, such metrics do not distinguish between a model that "misses" a relevant item by ranking it at position $K+1$ i.e., missing by just one place, and a model that ranks the relevant item last, after all the other items in the catalog. Obviously, the former model is better than the latter. In contrast, the MPR metric reports the average percentile rank of the relevant item with respect to all other items in the catalog. Ideally, the model would place the relevant items at the first percentile, hence a lower MPR indicates better results.

4.4 Results

In this section, we present the results of our experiments, including Hit@K, nDCG@K, Precision@K, Recall@K for various values of K, as well as the **Mean Percent Rank** (**MPR**) results for all datasets.

Tables 3–6 provide an overview of our model's performance in comparison to state-of-the-art path-based alternatives, such as RKGE and KPRN, across different evaluation metrics and values of K. The MPR results are summarized in Table 7. Notably, KPAR consistently outperforms these alternatives. Moreover, we compare KPAR with KGIN, a state-of-the-art propagation model, and demonstrate that KPAR achieves superior results.

In the following sections, we delve deeper into the components of KPAR to gain a better understanding of each component's contribution to the overall accuracy and performance of the model.

4.5 Ablation Study

As previously mentioned, the KPRN-Transformer model is based on the KPRN model of [46] with substituting the LSTM component with the self-attention path embedding component of KPAR.

Table 5. Comparison of Different Methods on the Precision@k Metric Across MovieLens, KKBox, and LastFM Datasets

	MovieLens - Precision@k				KKBox - Precision@k				LastFM - Precision@k			
Method	@1	@5	@10	@20	@1	@5	@10	@20	@1	@5	@10	@20
RKGE	0.268	0.222	0.201	0.178	0.462	0.387	0.341	0.292	0.085	*0.171*	0.086	0.044
KPRN	0.266	0.205	0.173	0.141	0.512	0.401	0.336	0.272	0.081	0.100	0.098	0.091
KPRN-Trans.	0.279	0.218	0.181	0.145	*0.587*	*0.489*	*0.428*	*0.350*	*0.171*	0.151	*0.135*	*0.112*
KGIN	*0.395*	*0.342*	*0.301*	*0.257*	0.172	0.150	0.134	0.117	0.081	0.062	0.049	0.038
KPAR	**0.498**	**0.382**	**0.314**	**0.242**	**0.643**	**0.538**	**0.466**	**0.380**	**0.184**	**0.254**	**0.251**	**0.229**

The best result is highlighted in **bold** and the second best is highlighted in *italics*.

Table 6. Comparison of Different Methods on the recall@k Metric Across MovieLens, KKBox, and LastFM Datasets

	MovieLens - Recall@k				KKBox - Recall@k				LastFM - Recall@k			
Method	@1	@5	@10	@20	@1	@5	@10	@20	@1	@5	@10	@20
RKGE	**0.037**	*0.084*	0.131	0.204	0.046	0.156	0.233	0.328	0.007	*0.070*	*0.072*	0.074
KPRN	0.019	0.073	0.123	0.197	0.054	0.168	0.247	0.343	0.005	0.029	0.052	0.087
KPRN-Trans.	0.021	0.079	0.129	0.202	*0.064*	*0.197*	*0.284*	*0.389*	0.010	0.040	0.067	*0.102*
KGIN	0.019	0.079	*0.133*	*0.217*	0.011	0.043	0.070	0.113	**0.014**	0.048	0.069	0.095
KPAR	*0.036*	**0.134**	**0.215**	**0.320**	**0.065**	**0.207**	**0.303**	**0.415**	**0.014**	**0.082**	**0.145**	**0.236**

The best result is highlighted in **bold** and the second best is highlighted in *italics*.

Table 7. MPR Results for MI, KKBox, and Last-FM (Lower is Better)

Dataset	KPAR	KPRN	KPRN-Trans.	RKGE
MI	**9.01%**	13.33%	14.25%	15.12%
KKBox	**7.22%**	8.85%	7.52%	10.60%
Last-FM	**10.5**%	23.4%	19.63%	45.7%

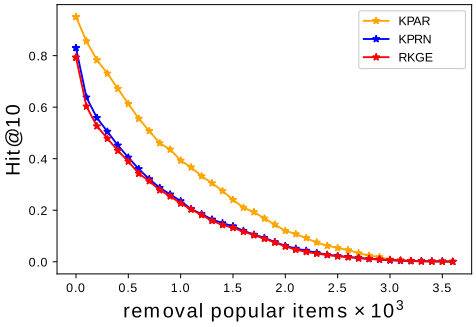
p-value < 0.05.

It is important to note that two notable distinctions exist between our KPAR model and KPRN. Firstly, whereas KPRN learns path embeddings using an LSTM module, KPAR acquires path representations through self-attention. Secondly, while KPRN aggregates path scores via logarithmic-average pooling, KPAR aggregates them using a parameterized cross-attention unit. As such, the KPRN-Transformer model can be seen as an ablated variant of KPAR which demonstrates the relative contribution of the model's components: By comparing KPRN-Transformer with KPRN, we can isolate the relative contribution of using self-attention for path representations. Additionally, by comparing KPRN-Transformer to KPAR we can isolate the relative contribution of KPAR's path aggregation obtained through cross-attention as opposed to pooling.

The results in Tables 3–6 and in Table 7 clearly indicate that aggregating path representations via self-attention is superior to using LSTM. This trend is very noticeable in the KKBox dataset. In the case of the MI dataset, the difference is less noticeable and somewhat mixed: KPRN-Transformer is better than KPRN on the Precision@K, Recall@K, and nDCG@K metrics, but sometimes lose to KPRN on the Hit@K and on the MPR metrics.

Comparing the KPRN-Transformer with KPAR, we see a clear advantage for KPAR across all metrics and datasets. This indicates the strong contribution of the cross-attention component in KPAR to its overall superior results. Hence, we conclude from this ablation study that both neural-attention components seem to contribute to the KPAR model, however, the cross-attention component, which learns to aggregate path representation with respect to their importance to the

(a) Hit@10 vs removed popular items on the MI dataset



(b) Hit@10 vs removed popular items on the KKBox dataset

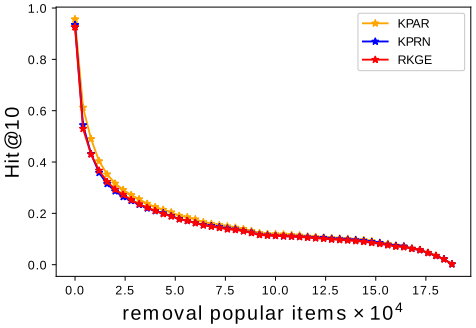


Fig. 3. Hit@10 vs removed popular items on the MI and the KKBox datasets.

final user-item prediction task, is considerably more significant. This insight teaches us that "not all paths are created equal" with respect to the user-item prediction task. By dynamically assigning attention scores to different path representations, KPAR can discern relevant and informative paths from non-informative ones. In Section 4.7 we go deeper into this and show how attention scores analysis can be utilized for better interpretability and explanations.

4.6 Cold Items

One distinctive advantage of KG-based recommender systems, when compared to classic CF methods, lies in their capability to leverage knowledge from the KG when transactional data is limited. To assess the models' ability to extract information from the KG in such scenarios, we conducted an additional evaluation. Specifically, we measured the Hit@10 metric while excluding the most popular items, placing greater emphasis on less popular items that rely more on KG data.

The objective of this evaluation was to determine if the model's performance remains robust when popular items are excluded. If the model performs well in this scenario, it suggests that it is less influenced by the popularity of items and can effectively utilize the KG. The results of this evaluation are presented in Figure 3, where we observe that KPAR maintains a significant lead over all baselines even as the number of popular items decreases. This underscores the model's proficiency in harnessing KG information.

To further expand our exploration, we investigate the model's performance when dealing with entirely cold items.

Pure CF methods are ill-suited for recommending completely cold itemsâ€"items without any transactional history. In contrast, KG-based recommender methods are hybrid models that combine CF data with KG data. Consequently, KG recommender methods can tap into KG information to recommend entirely cold items, demonstrating their versatility.

To evaluate the model's capability to recommend completely cold items, we conduct an analysis focused on evaluating results for these items. In our case, cold items are those connected solely to *persons* (e.g., directors, actors) via edges originating from the KG graph $\mathcal{KG}$ but not to *users* via edges from the CF graph $\mathcal{CF}$ (as defined in Section 3.1). To evaluate such cold items, we curated a dataset where test items have user connectivity exclusively in the test set, without any user interactions in the training dataset.

We perform this evaluation on the MI dataset, and the results are depicted in Figure 4. Once again, KPAR outperforms all other path-based baselines, emphasizing the model's superiority in effectively leveraging KG information, even for entirely cold items.

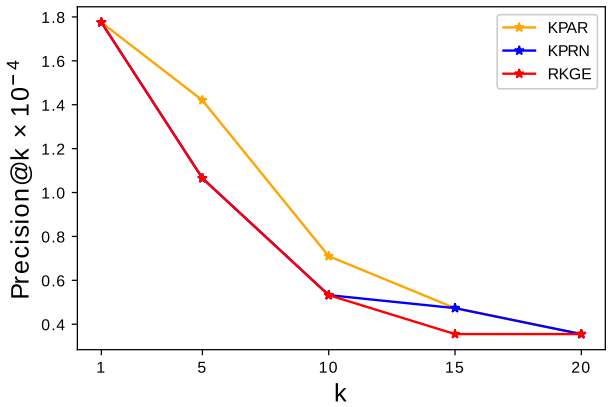


Fig. 4. Evaluation on completely cold items: Top-K performance between all baselines on the MI dataset, including cold items

4.7 Explainability

A key advantage of path-based KG methods over other KG methods stems from their ability to provide better interpretability and explanations. By following the paths between the user and the recommended item, interpretability for the model's decisions may be achieved and explanations can be generated. In the case of KPAR, the ability to explain recommendations is further improved thanks to its cross-attention mechanism which provides attentive scores over the paths that indicate their relative importance with respect to the recommendation (Section 3.2.3).

Figure 5 demonstrates KPAR's ability to provide interpretability and potential explanations on three recommendations to different users. Figure 5(a) depicts the top three paths which got the highest attention scores with respect to a recommendation to the movie "A Bug's Life (1998)". The path at the top connects the user and the item through the writer Joe Ranft who also wrote "Toy Story (1995)". This path received the highest score and arguably, it provides an intuitive explanation for the recommendation. The rest of the explaining paths connect the user to other users with a similar taste who also watched "A Bug's Life". These paths may be less useful for providing explanations, but they do provide interpretability into the inner workings of the KPAR model.

Figure 5(b) depicts the top three explaining paths to a user recommendation to the movie "Striking Distance (1993)". The top two explaining paths traverse through the actor "Bruce Willis" who played a leading role in the recommended movie as well as other movies the user has watched. The third recommendation is based on another user in the system with a similar taste.

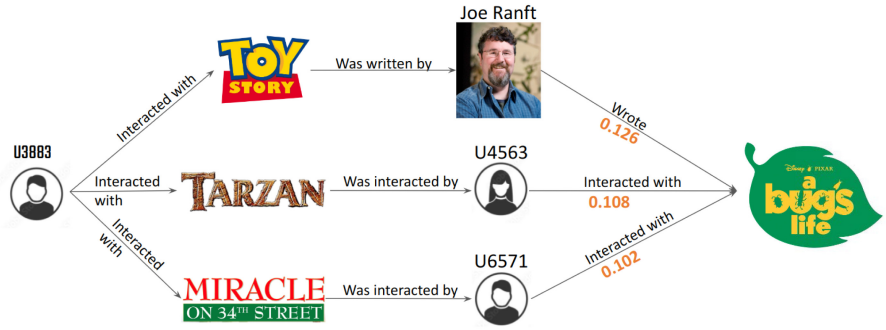
Figure 5(c) depicts the top three explaining paths to a recommendation of the movie "Peter Pan (1953)". The first two explaining paths traverse through the directors of "Peter Pan (1953)", who also directed other movies the user has watched i.e., "Alice in Wonderland (1951)" and "Dumbo (1941)". The third explaining path goes through another user with a similar taste who watched "The King and I (1999)".

The examples in Figure 5 illustrate the ease with which interpretability is achieved through attention score analysis in KPAR. Based on the most prominent paths, it is possible to generate intuitive explanations for users.

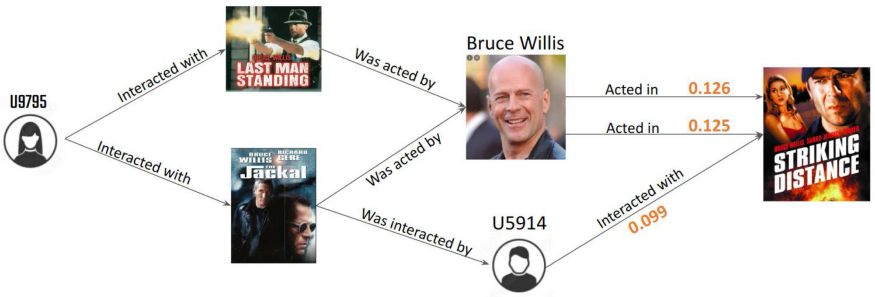
4.8 Item-to-Item Similarities

KPAR can also be utilized in order to extract item-to-item similarities. This is achieved by employing the model on paths that connect two items instead of paths that connect users to items. Figure 6 presents examples of item-to-item similarities extracted from the KPAR model. Similar to the

(a) Explaining a recommendation for the movie: “A Bug’s Life (1998)”.



(b) Explaining a recommendation for the movie: “Striking Distance (1993)”.



(c) Explaining a recommendation for the movie: “Peter Pan (1953)”.

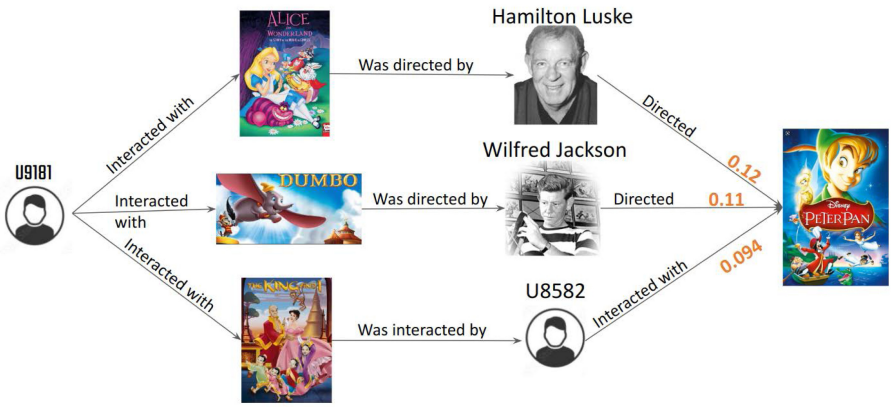
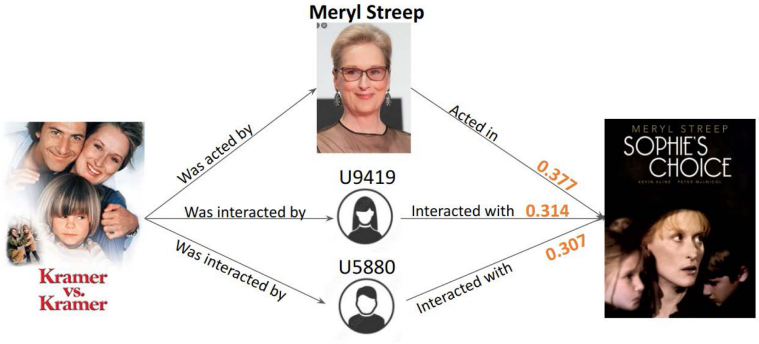


Fig. 5. Providing explanations and interpretability through attention weight analysis.

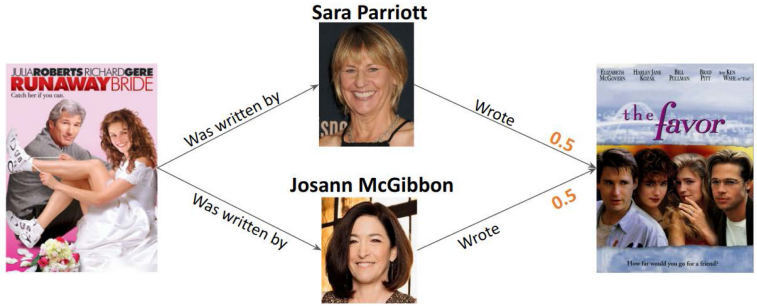
explanations of user-to-item recommendations, these similarities are easy to explain by following the most important paths.

Figure 6(a) depicts the paths which connect “Kramer vs. Kramer (1979)” with “Sophie’s Choice (1982)”. The path with the highest score traversed through the actress Maryl Streep who acted in both movies.

(a) Explaining the similarity of “Kramer vs. Kramer (1979)” to “Sophie’s Choice (1982)”.



(b) Explaining the similarity of “Runaway Bride (1999)” to “The Favor (1994)”.



(c) Explaining the similarity of “Conquest of The Planet of The Apes (1972)” to “Battle for the Planet of The Apes (1973)”.

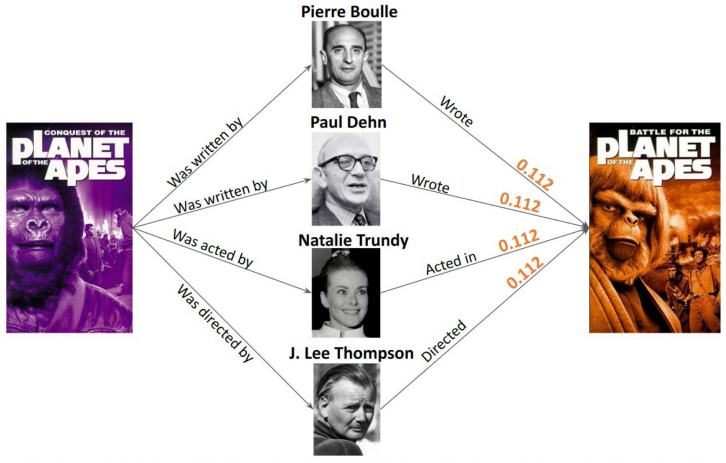


Fig. 6. Explaining Item to Item Similarities with KPAR.

Figure 6(b) presents a similarity between "Runaway Bride (1999)" and "The Favor (1994)". Interestingly, in this example, the movies are connected via two paths with equal scores. This similarity is explained by the mutual writers of both movies: Sara Parriott and Josann McGibbon which yields an equal score to both.

Figure 6(c) depicts the similarity between "Conquest of The Planet of The Apes (1972)" and "Battle for the Planet of The Apes (1973)", which are both sequels to the well-known movie "Planet of the Apes (1968)". The top 4 paths explain the similarity by exposing the common writers, a common actor, and a common director of both films. The score of these paths is similar since all paths contribute equally with respect to the similarity between the items.

5 Conclusion and Future Work

We presented KPAR - **K**nowledge-aware **P**ath **A**ttentive **R**ecommender, a novel path-based KG recommender. Different from state-of-the-art path-based methods that rely on RNNs, KPAR employs neural attention in order to represent paths on the KG. Furthermore, KPAR employs a novel path aggregation mechanism using cross-attention on the path representations which decides how much "attention" each path should receive with respect to a specific user-item scoring task. Using publicly available KG recommendation datasets, we show that KPAR presents superior results to recently published path-based KG recommendation algorithms. Furthermore, we demonstrate the ability of KPAR to naturally extract intuitive interpretability into its inner workings.

Providing explanations is a key advantage of path-based models [13]. In this work, we focused mainly on presenting the KPAR model itself and showcasing its different features. In future work, we plan to further investigate how to best extract useful explanations to users in different contexts and scenarios and in natural language. Though the presented explanations may serve developers of the system for better model understanding, we will investigate further how to adjust them to an end user.

References

[1] Behnoush Abdollahi and Olfa Nasraoui. 2016. Explainable matrix factorization for collaborative filtering. In *Proceedings of the 25th International Conference Companion on World Wide Web*. 5–6.

[2] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms* 11, 9 (2018), 137. DOI: https://doi.org/10.3390/a11090137

[3] Oren Barkan, Veronika Bogina, Liya Gurevitch, Yuval Asher, and Noam Koenigstein. 2024. A counterfactual framework for learning and evaluating explanations for recommender systems. In *Proceedings of the ACM on Web Conference 2024*. 3723–3733.

[4] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. 2011. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *Proceedings of the 5th ACM Conference on Recommender Systems*. 333–336.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems* 24 (2011), 2546–2554.

[6] Xianshuai Cao, Yuliang Shi, Jihu Wang, Han Yu, Xinjun Wang, and Zhongmin Yan. 2022. Cross-modal knowledge graph contrastive learning for machine learning method recommendation. In *Proceedings of the 30th ACM International Conference on Multimedia*. 3694–3702.

[7] Rose Catherine and William Cohen. 2016. Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 325–332.

[8] Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. 2021. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology* 12, 5 (2021), 1–32. DOI: https://doi.org/10.1145/3465055

[9] Yian Chen, Xing Xie, Shou-De Lin, and Arden Chiu. 2018. Wsdm cup 2018: Music recommendation and churn prediction. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 8–9.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An

image is worth 16x16 words: Transformers for image recognition at scale. arXiv:2010.11929. Retrieved from https://arxiv.org/abs/2010.11929

[12] Lisa Ehrlinger and Wolfram Wöß. 2016. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)* 48, 1–4 (2016), 2.

[13] Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. 2020. A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 35490–3568.

[14] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao. 2022. A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 1 (2022), 87–110.

[15] F. Maxwell Harper and Joseph A. Konstan. 2015. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (2015), 19 pages. DOI: https://doi.org/10.1145/2827872

[16] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 1661–1670.

[17] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. 173–182.

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735

[20] Dichao Hu. 2019. An introductory survey on attention mechanisms in NLP problems. In *Proceedings of the SAI Intelligent Systems Conference*. Springer, 432–448.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic gradient descent. In *International Conference on Learning Representations (ICLR'15)*. 1–15. Retrieved from https://arxiv.org/abs/1412.6980

[22] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR'17)*. Retrieved from https://arxiv.org/abs/1609.02907

[23] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[24] Charles Eric Leiserson, Ronald L. Rivest, Thomas H. Cormen, and Clifford Stein. 1994. *Introduction to Algorithms*. Vol. 3. MIT Press Cambridge, MA, USA.

[25] Andriy Mnih and Russ R. Salakhutdinov. 2008. Probabilistic matrix factorization. In *Proceedings of the Advances in Neural Information Processing Systems*. 1257–1264.

[26] Ingrid Nunes and Dietmar Jannach. 2017. A systematic review and taxonomy of explanations in decision support and recommender systems. *User Modeling and User-Adapted Interaction* 27, 3–5 (2017), 393–444.

[27] Enrico Palumbo, Diego Monti, Giuseppe Rizzo, Raphaël Troncy, and Elena Baralis. 2020. entity2rec: Property specific knowledge graph embeddings for item recommendation. *Expert Systems with Applications* 151, 113 (2020), 113235. DOI: https://doi.org/10.1016/j.eswa.2020.113235

[28] Tuan-Anh Nguyen Pham, Xutao Li, Gao Cong, and Zhenjie Zhang. 2016. A general recommendation model for heterogeneous networks. *IEEE Transactions on Knowledge and Data Engineering* 28, 12 (2016), 3140–3153. DOI: https://doi.org/10.1109/TKDE.2016.2601091

[29] Pearl Pu, Li Chen, and Rong Hu. 2011. A user-centric evaluation framework for recommender systems. In *Proceedings of the 5th ACM Conference on Recommender Systems*. 157–164.

[30] Steffen Rendle. 2010. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI'09)*. Retrieved from https://arxiv.org/abs/1205.2618

[32] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Proceedings of the Recommender Systems Handbook*. 1–35.

[33] Amit Singhal. 2012. Introducing the knowledge graph: Things, not strings. Retrieved 24-June-2024 from https://blog.google/products/search/introducing-knowledge-graph-things-not/

[34] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: A structural analysis approach. *Acm Sigkdd Explorations Newsletter* 14, 2 (2013), 20–28. DOI: https://doi.org/10.1145/2481244.2481248

[35] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003. DOI: https://doi.org/10.14778/3402707.3402736

[36] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. 2018. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 297–305.

[37] Ilaria Tiddi and Stefan Schlobach. 2022. Knowledge graphs as tools for explainable machine learning: A survey. *Artificial Intelligence* 302 (2022), 103627. DOI: https://doi.org/10.1016/j.artint.2021.103627

[38] Nava Tintarev and Judith Masthoff. 2007. A survey of explanations in recommender systems. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*. IEEE, 801–810.

[39] Nava Tintarev and Judith Masthoff. 2011. Designing and evaluating explanations for recommender systems. In *Proceedings of the Recommender Systems Handbook*. 479–510.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems*. 5998–6008.

[41] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 417–426.

[42] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 968–977.

[43] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *Proceedings of the World Wide Web Conference*. 3307–3313.

[44] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 950–958.

[45] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the Web Conference 2021*. 878–887.

[46] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 5329–5336.

[47] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. 2020. *CKAN: Collaborative Knowledge-Aware Attentive Network for Recommender Systems*. Association for Computing Machinery, New York, NY, USA, 219–228.

[48] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. 2020. CKAN: Collaborative knowledge-aware attentive network for recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 219–228.

[49] Yuhao Yang, Chao Huang, Lianghao Xia, and Chunzhen Huang. 2023. Knowledge graph self-supervised rationalization for recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3046–3056.

[50] Yuhao Yang, Chao Huang, Lianghao Xia, and Chenliang Li. 2022. Knowledge graph contrastive learning for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1434–1443.

[51] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. 283–292.

[52] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. 2013. Recommendation in heterogeneous information networks with implicit user feedback. In *Proceedings of the 7th ACM Conference on Recommender Systems*. 347–350.

[53] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 353–362.

[54] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 635–644.

[55] Ding Zou, Wei Wei, Xian-Ling Mao, Ziyang Wang, Minghui Qiu, Feida Zhu, and Xin Cao. 2022. Multi-level cross-view contrastive learning for knowledge-aware recommender system. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1358–1368.

Received 17 August 2023; revised 24 March 2024; accepted 14 May 2024