# Together is *Way* Better with Graph Neural Networks

Pasquale De Marinis
p.demarinis6@studenti.uniba.it

Lorenzo Loconte
l.loconte5@studenti.uniba.it

## Contents

# 1 Recommender Systems

In general, recommender systems are based on interactions between users, items and, optionally, static features. Recommender systems infer users' preferences about items and try to suggest relevant items to users.

The research branch of recommender systems can be divided in two fields based on the type of tasks (Wu et al., 2020).

1. **General recommendation** assumes that users have static preferences. In this case, we aim to predict users' interactions with items by using representations of both users and items.

2. **Sequential recommendation** assumes that users have dynamic (also evolving) preferences. In this case, we aim to predict the successive items that a user is likely to interact with, based on historical interactions of that user.

In this work, we will focus on the general recommendation task. Therefore, the next section will introduce a formal description of the task.

## 1.1 General Recommendation

Let $\mathcal{U}$ and $\mathcal{I}$ be the sets of users and items. Moreover, let $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{I}$ be a set of observed interactions between users and items. For any user $u \in \mathcal{U}$ and item $i \in \mathcal{I}$, the general recommendation task aims to estimate the preference of $u$ over $i$. This is done by learning user and item representations $h_u$, $h_i$. Finally, a score function $f(\cdot)$ is used to obtain a *degree of preference* of user $u$ on item $i$.

$$y_{u,i} = f(h_u, h_i) \tag{1}$$

The degree of preference $y_{u,i}$ is usually presented as the probability that the item $i$ is relevant for the user $u$. The definition of the score function $f(\cdot)$ is dependant on the representations of users and items. However, in the case of representations being vectors, the score function can be any linear or non-linear function, e.g. dot product, cosine similarity, neural network.

Most early studies considered user-item interactions in a matrix form, hence formulating the recommendation as a matrix completion task. Moreover, a lot of works focused on matrix factorization techniques that projects users and items to a Hilbert space, and then reconstructs the whole user-item interactions matrix, hence estimating unobserved users' preferences.

Recently, deep learning for recommender systems gained a lot of interest. That is because of the capability of deep neural networks to learn expressive embedding representations of users and items. However, major drawbacks of deep learning techniques include the general lack of explainability and the requirement of computational resources.

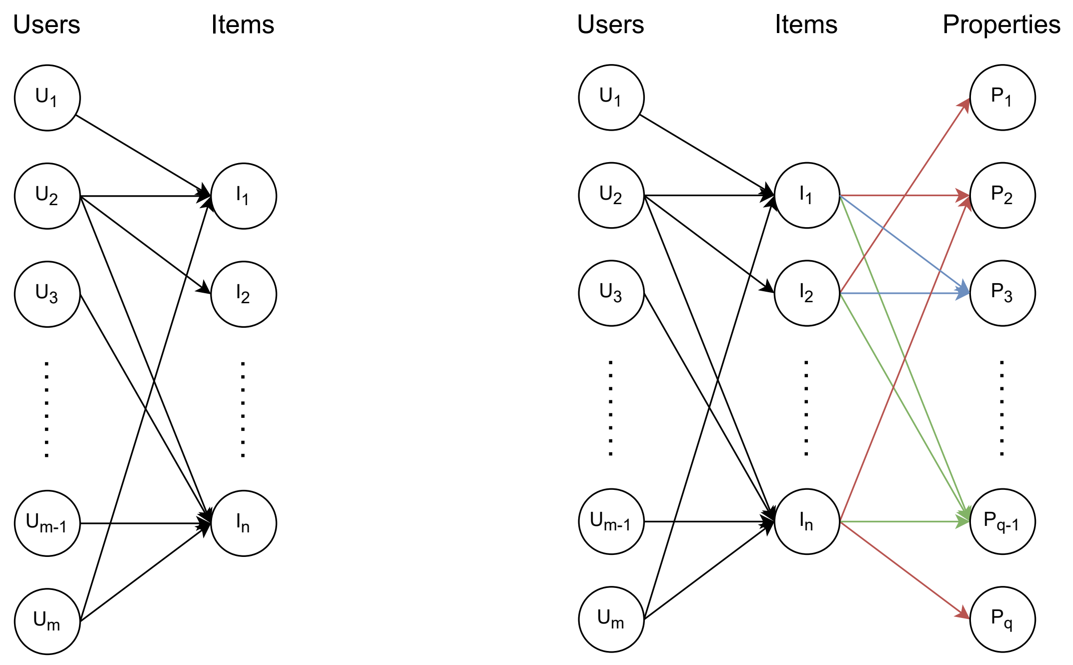

Figure 1: User-item interactions bipartite graph (on the left) and user-item-properties tripartite graph (on the right). Note that we have several relation types for the item-properties sub-graph (showed with different colors).

## 1.2 Graph Topologies for Recommendation

### 1.2.1 User-Item Graph

Interaction between users and items can be described as a graph, in particular, as a *bipartite graph*. A bipartite graph or *bigraph* is a graph $G = (V, E)$ such that:

$$V = V_1 \cup V_2 \tag{2}$$

$$E \subseteq V_1 \times V_2 \cup V_2 \times V_1 \tag{3}$$

Thus, any node in $V_1$ can be connected only with a node in $V_2$ and vice-versa. In a recommendation setting, $V_1$ and $V_2$ are respectively the set of users $\mathcal{U}$ and the set of items $\mathcal{I}$.

In a bipartite user-item graph, the interactions between two users can be implicitly represented by the commonly rated items. Conversely, the interactions between two items can be implicitly represented by the intersection of users that found such items relevant. These kinds of interactions are also called *higher-order*, and can be exploited easily to obtain additional information from a user-item graph.

### 1.2.2 Enhancing the User-Item Graph

It is possible to enhance the user-item graph by adding is present, a **knowledge graph** regarding the items, that express properties and relations between items. Incorporating knowledge graph into recommendation can bring two-facet benefits (Wang et al., 2019e):

- the rich semantic relatedness among items in a knowledge graph can help explore their connections and improve the item representation;
- knowledge graph connects a user's historically interacted items and recommended items, which enhances the interpretability of the results.

By adding a knowledge graph about properties of items, we will obtain what is called the *user-item-properties* graph. Since properties nodes are connected only to items, this graph is *tripartite*. Instead, in some cases, a complete knowledge graph can be attached to the user-item graph.

Another way to enrich the user-item graph can be done with the usage of *social networks*. Social recommender systems have been proposed to utilize each user's local neighbors' preferences to enhance user modeling. They assume users with social relationship should have similar representations, based on the social influence theory that connected people would influence each other. However, this is not the focus of this work.

Figures 1a and 1b show examples of a user-item graph and a user-item-properties graph respectively.

### 1.3 Task and Research Questions

In this work, we aim to find out how Graph Neural Networks (GNNs) literature is useful for our task of recommendation using deep neural networks. Specifically, we are interested in exploring the applications of GNNs in a hybrid recommender system, which is based on both collaborative and content-based extracted features (Polignano et al., 2021). For this purpose, we investigate the following research questions:

**RQ1** How well GNNs perform when used as collaborative features extractors in contrast to using pre-computed embeddings obtained by relying on Knowledge Graph Embeddings (KGEs) literature ?

**RQ2** How can we effectively integrate GNNs in a hybrid recommender system which leverages both collaborative and content-based features, i.e. also including both textual content and properties of items ?

As an additional, but less relevant task, we will expose a tentative of improving the network architecture of the hybrid recommender system by means of an attention mechanism and residual connections.

In Section 2 we will introduce a plethora of GNN models. Moreover, we will investigate them in relation to our task. In Section 5 we aim to respond to the introduced questions by means of empirical results about the introduction of GNNs in existing architectures.

| Base Framework | Model | Year | Properties | Implementations |
|---|---|---|---|---|
| GCN | GC-MC | 2018 | mean-pooling | Tensorflow1,Python2 |
| GraphSage | PinSage | 2018 | importance-pooling | PyTorch |
| GCN | SpectralCF | 2018 |  | Tensorflow1 |
| GCN | STAR-GCN | 2019 | mean-pooling | † |
| GraphSage | NGCF | 2019 | node affinity | Tensorflow1 |
| GraphSage | Bi-HGNN | 2019 |  | † |
| GCN | LR-GCCF | 2020 | w/o activation | PyTorch |
| GCN | LightGCN | 2020 | w/o activation & transformation | Tensorflow1, PyTorch |
| GraphSage | IG-MC | 2020 | sum updater | PyTorch, PyTorch-Geometric |
| GCN | DGCN-BinCF | 2019 | CrossNet | † |
| GCN | HashGNN | 2020 | mean-pooling | † |
| GraphSage | NIA-GCN | 2020 | neighbor interaction | † |
| GCN | AGCN | 2020 | w/o activation | † |
| hiearchical aggregation | MBGCN | 2020 |  | † |
| GAT | MCCF | 2020 |  | PyTorch |
| hiearchical aggregation | DGCF | 2020 |  | Tensorflow1 |
| GraphSage | GraphSAIL | 2020 |  | † |
| hiearchical aggregation | DisenHAN | 2020 |  | † |
| GAT | Gemini | 2020 |  | † |
| GraphSage | Multi-GCCF | 2019 |  | † |
| GCN | DGCF | 2020 | w/o activation & transformation | Tensorflow1 |
| GraphSage | TransGRec | 2020 | sum updater | † |

Table 1: Graph Neural Networks (GNNs) models suitable for user-item interactions. The *implementations* column contains links to Papers with Code pages with available implementations. † denotes that an existing implementation is not available at the time of this writing.

| Base Framework | Model | Graph | Year | Properties | Implementations |
|---|---|---|---|---|---|
| GAT | KGCN | KG | 2019 | | Tensorflow1 |
| GCN | KGNN-LS | KG | 2019 | user-specific adjacent matrix | Tensorflow1 |
| GAT | KGAT | Bi + KG | 2019 | | Tensorflow1 |
| GrahSage | IntentGC | U2U + I2I | 2019 | sum updater | TF-Euler * |
| GAT | AKGE | KG | 2019 | | † |
| GAT | MKGAT | Bi + KG | 2020 | | † |
| MetaPath | ACKRec | Bi + KG | 2020 | meta-path based | Tensorflow1 |
| GAT | ATBRG | KG | 2020 | | † |
| GAT | TGCN | Bi + KG | 2020 | hierarchical | † |

Table 2: Graph Neural Networks (GNNs) models suitable for several graph representations: bipartite (Bi), knowledge graph (KG), user-user (U2U) and item-item (I2I). The *implementation* column contains links to Papers with Code pages with available implementations. † denotes that an existing implementation is not available at the time of this writing.
* Custom framework with Chinese-only documentation.

## 2 Graph Neural Networks

In general, graph data is widely used to represent complex relationships and interactions of entities, e.g. users in a social graph. The success of deep learning techniques in the last years pushed researchers to analyze how such techniques can be used also for non-euclidean data, i.e. graph data.

Graph Neural Networks (GNNs) have demonstrated great performances and expressiveness on many tasks related to graph data, such as physical systems, protein structures, and knowledge graphs. The main idea of GNNs is to iteratively aggregate feature information from neighbors and integrate the aggregated information with the representations of central nodes. In other words, the representation of a node is updated by leveraging the representations of its neighbors in a propagation process (Wu et al., 2020). Note that, we will refer to *aggregation* as the process of collecting information in form of representations from the neighborhoods; *update* as the integration of the aggregated information with the representations of central nodes, hence updating nodes representations. The combination of aggregation and update processes is called *propagation*.

From a more practical perspective, multiple propagation layers can be stacked together in order to exploit non-trivial connections between nodes, hence permitting the nodes' information to *flow farther* in the graph. In order words, several propagation steps generally improve the entities' representation learning, since information of neighbors at different distances (also called *hops*) are considered as well. Moreover, there are various techniques for combining latent feature representations of entities obtained ad different layers. That is, most works either concatenate the embeddings obtained by each propagation step or average (possibly using weights) the embeddings. In the next sections, some of the GNN techniques will be presented, particularly for our task of collaborative and content-based items recommendation.

Table 1 and Table 2 show a representative collection of GNN models suitable for a bipartite graphs and for bipartite graphs connected to knowledge graphs respectively.

## 2.1 Graph Convolutional Networks

Let $\mathbf{A} \in \{0,1\}^{n_e \times n_e}$ be the adjacency matrix of a graph. Moreover, let $\mathbf{H}^{(l)} \in \mathbb{R}^{n_e \times d_h}$ denote the $l^{th}$ embeddings of each node having dimension $d_h$. Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) approximates the first-order eigendecomposition of the graph Laplacian to iteratively aggregate information from neighbors. In other words, it normalizes the adjacency matrix and then applies a non-linear transformation. The process of normalizing the adjacency matrix is also called *Laplace normalization*. Concretely, it updates the $l^{th}$ embedding $\mathbf{H}^{(l)}$ by

$$\mathbf{H}^{(l+1)} = \delta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{4}$$

where $\delta(\cdot)$ is the nonlinear activation function (e.g. $\mathrm{ReLU}$), $\mathbf{W}^{(l)}$ is the learnable transformation matrix for layer $l$; $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ where $A$ is the adjacency matrix of the undirected graph, $I$ the Identity used to add self-connections, and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$.

## 2.2 GraphSage

GraphSage (SAmple and aggreGatE) (Hamilton et al., 2017) samples a fixed size of neighborhood for each node. It then proposes mean/sum/max pooling aggregators and adopts concatenation operation for update. That is, given a sampled subset of neighbors $\mathcal{N}_v$ of node $v$, the representation of the central node $\mathbf{h}_v^{(l)}$ is updated as

$$\mathbf{n}_v^{(l)} = \mathrm{AGGREGATE}_l(\{\mathbf{h}_u^{(l)}, \forall u \in \mathcal{N}_v\}) \tag{5}$$

$$\mathbf{h}_v^{(l+1)} = \delta(\mathbf{W}^{(l)} \cdot [\mathbf{h}_v^{(l)} \oplus \mathbf{n}_v^{(l)}]) \tag{6}$$

where $\mathrm{AGGREGATE}_l$ denotes the aggregation function at $l^{th}$ layer, $\oplus$ denotes the vector concatenation, $\delta(\cdot)$ is the nonlinear activation function (e.g. $\mathrm{ReLU}$), and $\mathbf{W}^{(l)}$ is the learnable transformation matrix and $\mathbf{n}_v^{(l)}$ is the final node representation at the layer $l$.

## 2.3 Graph Attention Networks

Graph Attention Networks (GATs) (Velickovic et al., 2017) assumes that the influence of neighbors on the central node is neither identical nor pre-determined by the graph structure, thus it differentiates the contributions of neighbors by leveraging attention mechanism and updates the vector of each node by attending over its neighbors

$$\alpha_{vj} = \frac{\exp(\mathrm{LeakyReLU}(\mathbf{a}^T[\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \oplus \mathbf{W}^{(l)}\mathbf{h}_j^{(l)}]))}{\sum_{k \in \mathcal{N}_v} \exp(\mathrm{LeakyReLU}(\mathbf{a}^T[\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \oplus \mathbf{W}^{(l)}\mathbf{h}_k^{(l)}]))} \tag{7}$$

$$\mathbf{h}_v^{(l+1)} = \delta\left(\sum_{j \in \mathcal{N}_v} \alpha_{vj}\mathbf{W}^{(l)}\mathbf{h}_j^{(l)}\right) \tag{8}$$

where $\delta(\cdot)$ is the nonlinear activation function (e.g. $\mathrm{ReLU}$), $\oplus$ denotes the vector concatenation, $\mathbf{a}$ is a learnable parameter, and $\mathbf{W}^{(l)}$ is the learnable transformation matrix for layer $l$.

## 2.4 Knowledge Graph Attention Networks

Knowledge Graph Attention Networks (KGAT) (Wang et al., 2019c) are particularly useful in modeling interactions in a graph consisting of the user-item graph and the item-entity knowledge graph. That is, in addition to the user-item bipartite graph, we have side information for items in the form of a knowledge graph.

KGAT is based on GAT for the use of attention mechanisms in (multi-hop) propagation layers. However, the presence of a knowledge graph setting induces two different loss functions for the underlying graphs. So, the objective of KGAT is to minimize both a collaborative filtering loss based on user-item interactions (Equation 9), and a knowledge graph loss based on item-entity and entity-entity interactions (Equation 10).

$$\mathcal{L}_{\text{CF}} = \sum_{(u,i,j)\in\mathcal{O}} -\ln\sigma\left(\hat{y}(u,i) - \hat{y}(u,j)\right) \tag{9}$$

$$\mathcal{L}_{\text{KG}} = \sum_{(h,r,t,t')\in\mathcal{T}} -\ln\sigma\left(g(h,r,t) - g(h,r,t')\right) \tag{10}$$

where $\mathcal{O} = \{(u,i,j) \mid (u,i) \in \mathcal{R}^+, (u,j) \in \mathcal{R}^-\}$ denotes the training set, $\mathcal{R}^+$ the positive interactions, and $\mathcal{R}^-$ the (sampled) negative interactions.
$\mathcal{T} = \{(h,r,t,t') \mid (h,r,t) \in \mathcal{G}, (h,r,t') \notin \mathcal{G}\}$, denoting the knowledge graph as $\mathcal{G}$, and $(h,r,t')$ is a broken triplet constructed by replacing one entity in a valid triplet randomly.

Moreover, to fully capture the semantic information in a knowledge graph, KGAT also considers also the relation types. So, the idea is to differentiate in the propagation steps the weights of the neighbours based on their relation type. This is done by an attention modules that leverages a set of weights for each relation type.

## 2.5 Knowledge Graph Convolutional Networks

While some works simplify the graph structure or construct the a subgraph at first stage based on the information of knowledge graph and user-item bipartite graph, Knowledge Graph Convolutional Networks (KGCN) (Wang et al., 2019b) apply a GNN model over the original knowledge graph.

KGCN do *not* combine both the bipartite graph and the knowledge graph. That is, it explicitly characterize users' interest in specific relation types (e.g. the presence of an actor in a movie) by leveraging weights given by the interactions between a user and a relation type. Similarly to KGAT, this is done by making use of an attention module that weights differently the neighbours in the knowledge graph. That is, given two entities $e_i$, $e_j$, an user $u$ and a relation type $r_{e_i,e_j}$, we can define the attention score of the user $u$ with respect to the relation type connecting $r_{e_i,e_j}$ as, for example, the dot product of their embeddings.

$$a(e_i, e_j, r_{e_i,e_j}, u) = \mathbf{u}^T \mathbf{r}_{e_i,e_j} \tag{11}$$

In this way, entities whose relations are more consistent with users' interests will be more relevant, hence spreading more information in the propagation step.

## 2.6 Deoscillated Adaptive Graph Collaborative Filtering

The direct application of GNNs to a bipartite graph arises the *oscillation problem* (also called *varying locality problem*). For example, the user-item bipartite graph in a recommendation setting. The direct neighbors of users are all items, and the direct neighbors of items are all users. Moreover, neighbors at distance two of users are only users, and the same for items. This implies that by aggregating the direct neighbors, users only receive the information from items, and vice versa. So, it turns out that the information oscillates between users and items (Liu et al., 2020).

The idea of Deoscillated Adaptive Graph Collaborative Filtering (DGCF) (Liu et al., 2020) is to solve this problem by introducing *cross-hop* connections in a bipartite graph, i.e. including in the set of users/items neighbours both users and items, at each propagation step. The obtained adjacency matrices are normalized as for Graph Convolutional Networks (GCNs), i.e. the Laplacian normalization. Moreover, it uses trainable *locality weights* to each node at each propagation step, in order to differently weight nodes at the aggregation steps (without using attention mechanisms). Thus, the embedding update is done by:

$$\mathbf{H}^{(l+1)} = \mathbf{\Omega}^{(l)}((\mathcal{L} + \mathcal{L}_c + \mathbf{I}) \cdot \mathbf{H}^{(l)}) \tag{12}$$

Where $\mathcal{L}$ and $\mathcal{L}_c$ are the Laplacian normalized adjacency and cross-hop matrix and $\mathbf{\Omega}^{(l)}$ a diagonal matrix containing the locality weights of the $l$-th layer.

However, including cross-hop connections introduces extra computational costs. This can be easily solved by noticing that some entries in the Laplacian normalized adjacency matrix are usually very small values. So, we can *filter out* these values, hence reducing the density of the underlying graph itself.

## 2.7 Oversmoothing Problem

Before proceeding with a deep analysis of suitability of GNNs for our purposes, we will summarize here a well-known problem for all GNNs models: the *oversmoothing problem*.

In general, stacking several GNN layers in sequence permits to model higher-order dependencies between nodes. That is, the latent features representation of a node is given by not only its direct neighbors, but also by its neighbors at a distance more than one (i.e. multi-hop) in the underlying graph. So, the more GNN layers we stack in sequence, the more neighbors of a node will take part in the process of learning its latent representation, generally in the form of an embedding. Moreover, on average, the number of neighbors of a node grows exponentially with respect to the out-degree of the underlying graph and the distance from the node itself (i.e. the number of hops).

The *oversmoothing problem* arises from the fact that, with a relatively high number of GNN layers, nodes have approximately the same higher-order neighbors in common. Therefore, the learned latent representations of nodes will be very similar, hence not permitting to effectively differentiate the nodes. A metric suitable for measuring smoothness is the *mean average distance* (MAD) between pairs of node embeddings. The result of this problem is a significant drop in performances, which is exacerbated in the presence of a high out-degree factor (Chen et al., 2020a).

The *oversmoothing problem* can be relieved by either limiting the number of GNN layers (i.e. reducing the higher-order node interactions), or by applying a training regularization factor that penalizes very similar nodes' embeddings.

## 3 Models and Techniques Analysis

For the sake of simplicity and brevity, in this section, there will be illustrated only the properties and features of works that have shared at least one implementation. Moreover, we will analyze and discuss several Graph Neural Networks (GNNs) in relation to our tasks of recommendation.

Table 1 shows a comprehensive list of GNNs suitable for bipartite (Bi) graphs, i.e. user-item interactions. In addition, Table 2 shows a comprehensive list of GNNs also suitable for knowledge graphs (KG), user-user (U2U) and item-item (I2I) interactions.

### 3.1 Graph Convolutional Networks

The aggregation strategy of GCN, is, obviously based on graph convolution; however, some works to cope with the multi-type relations between users and items, design a hierarchical aggregation strategy and observe effectiveness gains.

For example, MBGCN (Jin et al., 2020) firstly aggregates the interacted items belonging to each behavior respectively (e.g. click, collect, purchase, share) and further integrates the different aggregated behaviors. However, in our case, the dataset contains only ratings, therefore these methods are not needed. With respect to the representation of nodes in LightGCN (He et al., 2020), GC-MC (Su et al., 2021), DGCF (Liu et al., 2020) use neighbors as representation of the node. However, this strategy might overlook the original user preference/item properties. Regarding the updating of information: LightGCN and LR-GCCF (Chen et al., 2020b) remove the non-linearities to increase efficiency, since the authors show it increases only a bit in the performances.

The final representation, so, the output in LR-GCCF and SpectralCF (Zheng et al., 2018) it is the concatenation of all layers. This emphasizes message passing through the layers and maintains original information about the node (first layers) and neighborhood information (final layers). DGCF instead uses mean pooling though layers as final representation and LightGCN uses a more refined strategy, that is layer-wise weighted pooling.

**For knowledge graphs**, two implementations are available: KGCN and KGNN-LS (Wang et al., 2019a). The first one have been already described in Section 2.5. The second one is a variant of KGCN, where LS stands for *label smoothing*. This model transforms the graph in a user-specific one by first applying a trainable function that identifies important knowledge graph relationships for a given user, then apply the GNN with *label smoothing* regularization; which assumes that that adjacent items in the knowledge graph are likely to have similar user relevance labels/scores.

### 3.2 GraphSage Implementations

Specific available implementations of GraphSage are PinSage (Ying et al., 2018), IG-MC (Zhang and Chen, 2020) and NGCF (Wang et al., 2019d). The aggregation is done through linear aggregation functions (e.g. mean or sum). NGCF employs element-wise product to augment the items' features the user cares about or the users' preferences for features the item has. Node update is done through concatenation and non-linearity (e.g. $\mathrm{LeakyReLU}$ or $\mathrm{ReLU}$) for PinSage, using a linear combination of node and neighbors aggregates for IG-MC and NCFG.

As final representation, PinSage uses only the last layer, in contrast with IG-MC and NGCF that use concatenation of all the layers' outputs. However, this strategy can cause the *oversmoothing problem*. Furthermore, PinSage and IG-MC have a sampling strategy suitable for very large graphs (e.g. with a million of nodes), that can randomly take far nodes neighbors too. Therefore, we discard PinSage and IG-MC as are suited for very large graphs, that is certainly not our case.

**For knowledge graphs**, a variant of GraphSage is given by IntentGC (Zhao et al., 2019), which is suitable for item-user graphs enhanced by knowledge information. Differently from KGCN, it performs a graph simplification by translating the first-order proximity in the multi-entity knowledge graph into second-order proximity, i.e. keeping the item-to-item relationship that only one node apart. Specifically, if items $i_1$ and $i_2$ are both connected by an auxiliary node (the user or the property), the type of the auxiliary node is denoted as the relationship between these two items.

The advantage of this transformation is that it turns the multi-entity graph into a homogeneous graph from the perspective of node types. Therefore, it ends up with an U2U graph and an I2I graph. However, this strategy is not fit for the graph where most of the item nodes are multi-hop (more than two-hop) neighbors, since it greatly simplifies the graph structure at the cost of loss of linked information between two item nodes. Furthermore, it requires different types of interactions between users and items. Therefore, it is not suitable for our dataset, since we have only positive and negative interactions between users and items.

## 3.3 Graph Attention Networks

Graph Attention Networks (GATs) have a unique aggregation strategy with respect to other models since they use an attention mechanism to properly weight each neighbor in the aggregation step.

In addition to the base model of GAT, the only available implementation is MCCF (Wang et al., 2020). It decomposes the nodes using different latent spaces that represents an aggregation of items (or users) and weights them using an attention mechanism; then subsequently recombines them. The attention mechanism is here used two times, on *node level* and on *component level*.

**For knowledge graphs**, we can notice in Table 2 that most of the models uses GAT as a base architecture. However, KGAT is the only one having a freely available implementation.

## 3.4 Proposed models

Since Graph Neural Networks (GNNs) models in literature for knowledge graphs did not satisfy our requirements, as they are intended to work with a full Knowledge Graph (KG) and not a simple tripartite graph; we managed to combine GNNs to work with a tripartite graph.

A possible approach is the direct application of GNN models to the user-item-properties tripartite graph. However, the simplicity and efficiency of this approach has important drawbacks. That is, it considers as neighbors of an item both users and

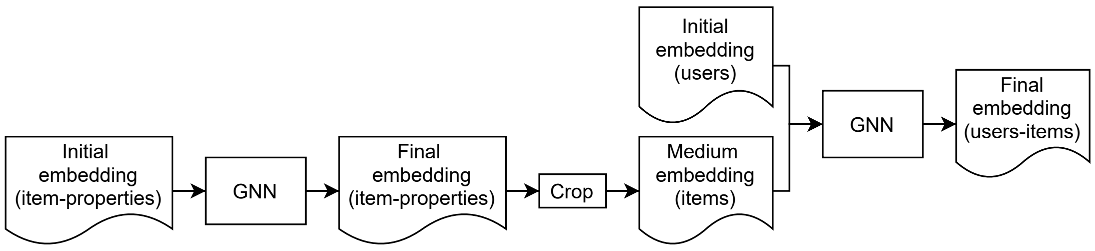
Initial embedding (item-properties)
GNN
Final embedding (item-properties)
Crop
Initial embedding (users)
Medium embedding (items)
GNN
Final embedding (users-items)

Figure 2: Two-Step Graph Neural Network (GNN). The embeddings of items are firstly obtained by a GNN model using the item-properties graph, and then passed through another GNN model using the user-item graph.

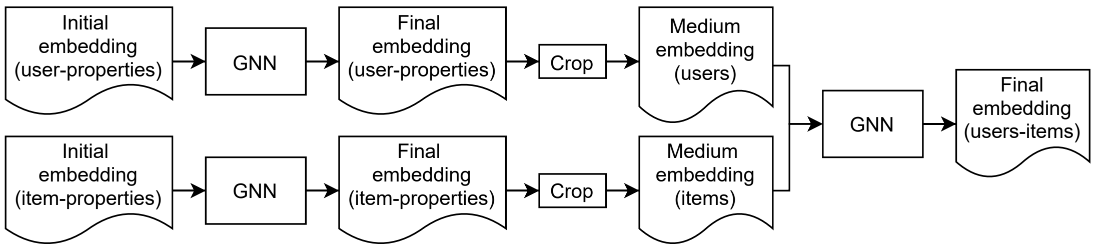
Initial embedding (user-properties)
GNN
Final embedding (user-properties)
Crop
Medium embedding (users)
Initial embedding (item-properties)
GNN
Final embedding (item-properties)
Crop
Medium embedding (items)
GNN
Final embedding (users-items)

Figure 3: Two-Way Graph Neural Network (GNN). Initial embeddings of users and items are firstly obtained by two GNN models using the user-properties and the item-properties graphs respectively. The resulting users and items embeddings are then passed through another GNN model using the user-item graph.

properties, and treat them in the same way. Moreover, the item-properties graph is generally much more sparse than the user-item bipartite graph. Therefore, we generally have an *asymmetry* of information that will be propagated in order to learn items' latent representations.

In order to solve these issues, two models based on GNNs will be described in the following sections: Two-Step GNN and Two-Way GNN. Note that in Section 5 we include empirical results about the naive approach of applying GNN models to the user-item-property graph, and the performances of both the Two-Step and Two-Way GNNs.

### 3.4.1 Two-Step GNN

In order to integrate the item-properties graph, we added another GNN that will extract embeddings of properties and items. Subsequently, the embedding properties are discarded, and the item ones will be fed into the second GNN. Thus, the KG information will be encoded into the item embeddings. The users' embeddings, instead, will be randomly initialized as usual and trained jointly with the rest of the model. The architecture, called Two-Step GNN, is summarized in Figure 2.

The embedding size is now constrained by the output of the first GNN, so by its final node representation (e.g. by average, concatenation, ...) and its initial embedding dimension. To avoid by ending up with a too large final embedding, the output of the first GNN, i.e. the one using the item-properties graph, will use the average operation to combine the embeddings at each propagation step. Thus, the users em-

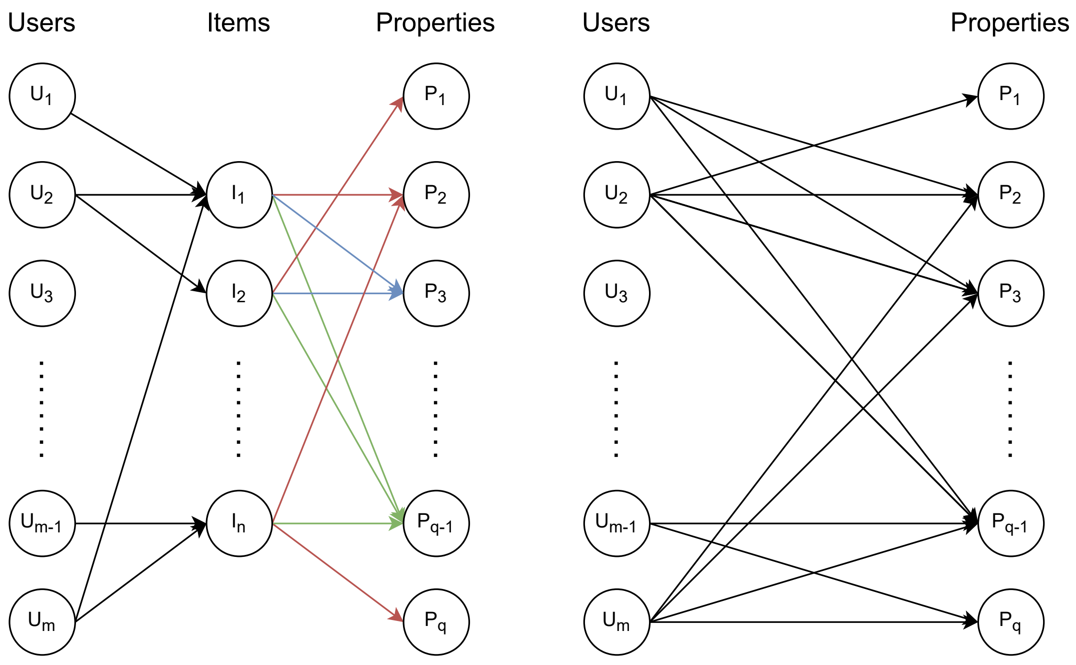

Figure 4: User-item-properties tripartite graph (on the left) and user-properties pre-processed bipartite graph (one the right). Note that there is an arc between a user and a property if there is a directed path of length two passing through an item in the user-item-properties graph.

beddings will have the same dimensionality of items embeddings.

The main idea of this architecture is that the initial latent representations of items are obtained based on the related properties. After that, they are further refined using also the latent representations of users.

### 3.4.2 Two-Way GNN

In the Two-Step GNN architecture, at the input time of the second step, the items' embeddings incorporated the information about the properties, while the users' ones are initialized at random and jointly trained. We try to avoid this asymmetry with the Two-Way GNN architecture, that has a branch for users too before the second step.

In order to feed some input to this additional branch, we built a user-properties adjacency matrix starting from the user-item and item-properties ones. This operation is made by first building the user-item-properties tripartite graph, and then removing each item node $i$, replacing it with a set of edges $(u, p)$ for each property $p$ and user $u$ connected to $i$. The process is illustrated in Figure 4. It is noticeable that the sparsity of the resulting user-properties matrix is reduced, since the average out-degree of the new bipartite graph is the product of the two input bipartite graphs. Finally, the overall architecture of Two-Way GNN is shown in Figure 3.

The main idea of this architecture is to learn users' latent representations also by taking into account the properties that a user implicitly deems relevant. Moreover, the items' latent representations are also learned in relation to the properties, as in the Two-Step GNN architecture just discussed.

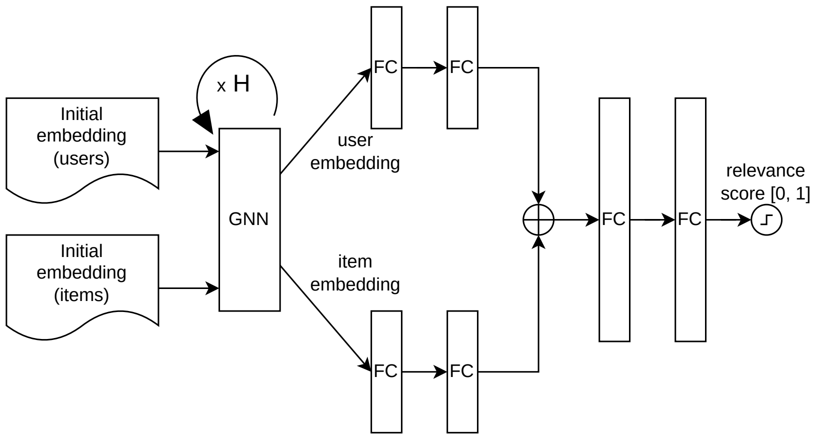

Figure 5: General BasicRS architecture with GNN models as embeddings extractor. Initial users and items embeddings are passed through multiple GNN propagation steps, by making use of neighbours information. The resulting embeddings are then used as features of the BasicRS model. The output of the entire network represents the likelihood that an item is relevant for a user.

## 4 Deep Recommender Systems with GNNs

### 4.1 Basic Architecture

In this section, we consider the *basic* neural network architecture introduced by Polignano et al. (2021), which has been extended by V. Digeno in a previous work[1]. We will refer to this model as BasicRS (Basic Recommender System).

BasicRS takes as inputs the features embedding vectors of users and items. After passing through several dense layers and a concatenation step, it produces a relevance score between 0 and 1, which represents the likelihood that an item is relevant or not for a user. In general, we can firstly learn the latent features representations of users and items, and then train such model to make predictions. In the original formulation, this is done by either relying on collaborative or content-based features. That is, we can exploit the user-item interactions graph to learn embeddings using Knowledge Graph Embedding (KGE) models. Alternatively, we can exploit features extracted from the items' textual content by a transformer (e.g. BERT), and then computing users embeddings by combining the embeddings of relevant items (e.g. the centroid).

In this work, we are going to replace the users and items preliminary embeddings learning step with a multi-hops Graph Neural Network (GNN), and train the entire model in an end-to-end fashion. So, the idea is to rely on the collaborative information given by the user-item interaction graph to effectively learn embeddings of users and items, which are fed to the BasicRS model. A graphical representation of the BasicRS architecture with GNN models is showed in Figure 5.

For simplicity, we refer to the BasicRS model using TransD features as BasicRS-TransD, and to the BasicRS model using a GCN as embedding extractor as BasicRS-GCN. The same nomenclature also applies for the other KGE and GNN models.

[1]https://github.com/cenzy/Deep_CBRS_Amar_Revisited

## 4.2 Hybrid Architectures

In this section we consider the *hybrid* neural network architectures introduced by Polignano et al. (2021), which has been extended similarly to BasicRS, i.e. by making it deeper. We will refer to this model as HybridCBRS (Hybrid Content-Based Recommender System).

For the BasicRS model, we can rely either on collaborative features or on content-based features. The HybridCBRS models are instead based on both collaborative features and content-based features. As already said, the collaborative features are the same used in the BasicRS model, i.e. pre-computed using Knowledge Graph Embeddings (KGE) models (e.g. TransD) using the user-item bipartite graph. The content-based features are computed starting from BERT word embeddings. That is, the latent features representations of items are defined as the sum of BERT embeddings of words appearing in the description of items (e.g. the plot for movies). Then the latent features representations of an user is defined as the centroid of items whose user expressed positive review.

In fact, there are two kinds of HybridCBRS models, based on the ways they concatenate collaborative and content-based features in the network's architecture. More formally, let $\mathbf{u}_{\text{graph}}$ and $\mathbf{u}_{\text{bert}}$ be collaborative and content-based embedding vectors of a user respectively; and let $\mathbf{v}_{\text{graph}}$ and $\mathbf{v}_{\text{bert}}$ be collaborative and content-based embedding vectors of an item respectively. The concatenation methods in HybridCBRS models are the following.

- **Entity-based**: collaborative and content-based features of users and items are concatenated by entity type:

$$\mathbf{x}_1 = \phi_1(\mathbf{u}_{\text{graph}} \oplus \mathbf{u}_{\text{bert}}) \tag{13}$$
$$\mathbf{x}_2 = \phi_2(\mathbf{v}_{\text{graph}} \oplus \mathbf{v}_{\text{bert}}) \tag{14}$$

- **Feature-based**: collaborative and content-based features of users and items are concatenated by features type:

$$\mathbf{x}_1 = \phi_1(\mathbf{u}_{\text{graph}} \oplus \mathbf{v}_{\text{graph}}) \tag{15}$$
$$\mathbf{x}_2 = \phi_2(\mathbf{u}_{\text{bert}} \oplus \mathbf{v}_{\text{bert}}) \tag{16}$$

where $\phi_1$ and $\phi_2$ denote dense networks with a non-linearity (e.g. $\mathrm{ReLU}$), and $\oplus$ denotes the concatenation operator. The obtained feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are again concatenated and passed through dense layers and used for relevance classification. However, since in the original work about the *hybrid* architecture (Polignano et al., 2021) the best results are obtained using the feature-based concatenation method, we rely on the same architecture.

In this work, we replace the pre-computed collaborative features with Graph Neural Networks (GNNs), similarly to the BasicRS model with GNNs described in the previous section. However, we will keep the pre-computed BERT embeddings of both users and items. Note that, we used BERT embeddings that are obtained without using stopwords.

A graphical representation of the HybridCBRS feature-based architecture with GNN models is showed in Figure 6.

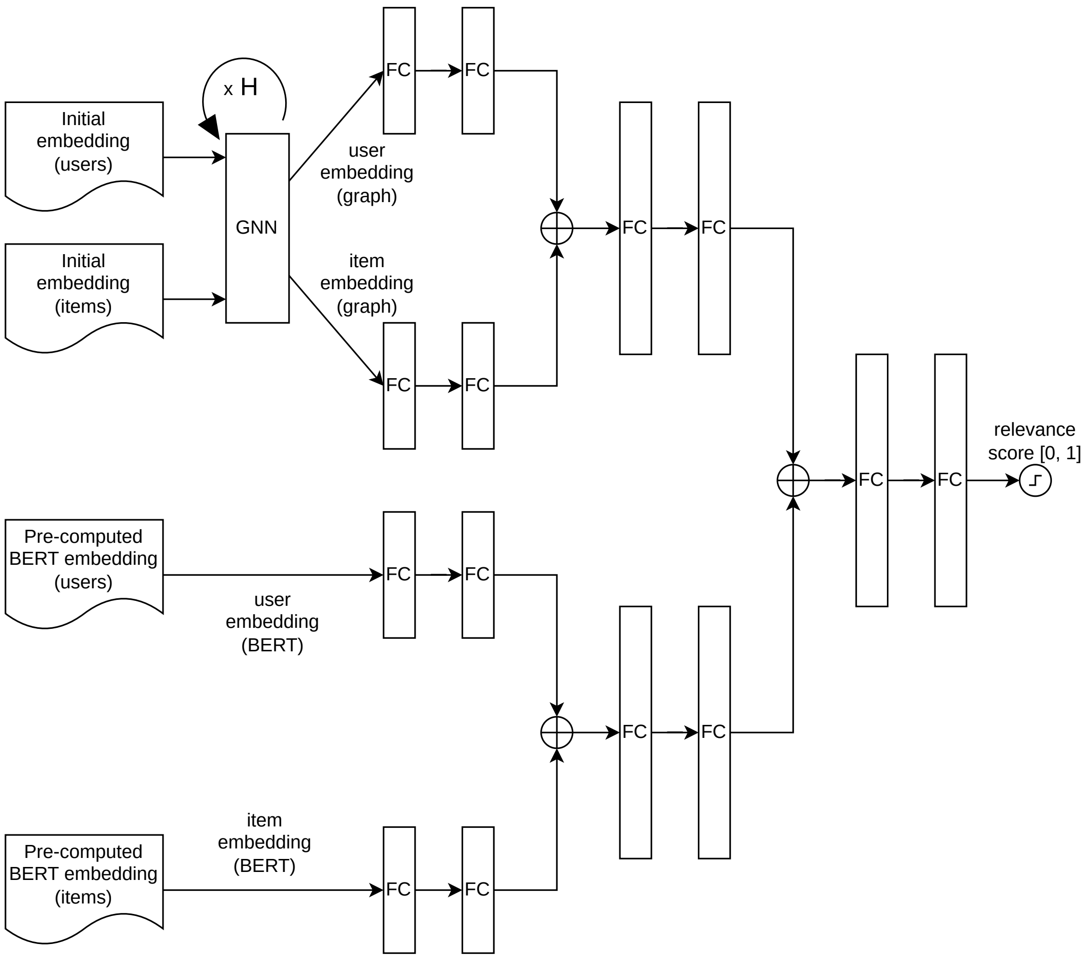
x H
Initial embedding (users)
Initial embedding (items)
GNN
user embedding (graph)
item embedding (graph)
FC
Pre-computed BERT embedding (users)
Pre-computed BERT embedding (items)
user embedding (BERT)
item embedding (BERT)
relevance score [0, 1]

Figure 6: General HybridCBRS feature-based architecture with GNN models as graph embeddings extractor. Initial users and items graph embeddings are passed through multiple GNN propagation steps, by making use of neighbours information. The resulting graph embeddings are then used as features for the graph-related branches of the HybridCBRS model. Moreover, pre-computed BERT embeddings are used for the other input branches. The output of the entire network represents the likelihood that an item is relevant for a user.

## 4.3 Hybrid Architecture Tweaks

In this section, we are going to present two tweaks for the feature-based *hybrid* architecture presented in the previous sections. These tweaks are about substituting the last concatenation operation.

### 4.3.1 Self-Attention Mechanism

The first tweak is based on the idea that, in the classification subnetwork of the feature-based HybridCBRS model, we might be interested in *differentiating* the relevance weights of the latent features coming from the two different network's branches (i.e. collaborative and content-based). In other words, is it possible that in some cases better predictions are given by considering more the features given by the content-based branch than the collaborative branch (or also vice-versa). This idea drives us a *self-attention mechanism* that differentiate features, which replaces the last concatenation step just before the user-item relevance prediction.

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be feature vectors obtained by the feature-based concatenation method showed in the previous section. We propose a self-attention mechanism given by the following equations.

$$\mathbf{s}_1 = \tanh(\mathbf{W}\mathbf{x}_1) \tag{17}$$

$$\mathbf{s}_2 = \tanh(\mathbf{W}\mathbf{x}_2) \tag{18}$$

$$a_{1i} = \frac{\exp(s_{1i})}{\exp(s_{1i}) + \exp(s_{2i})} \qquad \forall i \tag{19}$$

$$a_{2i} = \frac{\exp(s_{2i})}{\exp(s_{1i}) + \exp(s_{2i})} \qquad \forall i \tag{20}$$

$$\hat{\mathbf{x}} = \mathbf{a}_1 \odot \mathbf{x}_1 + \mathbf{a}_2 \odot \mathbf{x}_2 \tag{21}$$

where $\mathbf{W}$ is a square weight matrix, and $\odot$ denotes the element-wise product. Finally, instead of using the concatenation of $\mathbf{x}_1$ and $\mathbf{x}_2$ as features for the predictions, we propose $\hat{\mathbf{x}}$ as a weighted sum of $\mathbf{x}_1$ and $\mathbf{x}_2$ features.

### 4.3.2 Residual Connections

The second tweak is based on the fact thej collaborative and content-based subnetworks of the feature-based HybridCBRS model are identical to the BasicRS model (without the last layer for classification). The idea is to directly incorporate the outputs of these networks as additional features for the classification, i.e. using *residual connections* (He et al., 2016), which have been shown to improve training convergence.

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be feature vectors obtained by the feature-based concatenation method showed in the previous section. We propose residual connections given by the following equations.

$$\hat{\mathbf{x}} = \delta(\gamma(\mathbf{x}_1 \oplus \mathbf{x}_2) + \mathbf{x}_1 + \mathbf{x}_2)$$

$$\hat{y} = \sigma(\mathbf{w}^T \hat{\mathbf{x}} + b)$$

where $\gamma$ is a dense network whose last non-linearity have been removed, $\delta$ is a non-linearity (e.g. $\mathrm{ReLU}$), $\mathbf{w}$ and $b$ are respectively weight and bias of the last layer for classification, $\sigma$ denotes the sigmoid function, and $\oplus$ denotes the concatenation operator. Finally, $\hat{y} \in [0, 1]$ is the user-item relevance output.

## 5 Experiments

In this section, we will present all the experiments performed and show the relative results. Each set of experiment has its own table of parameter and results. However, to have an easier and faster comparison of the different settings, the best overall results are summarized in two tables: Table 15 for BasicRS and Table 16 for HybridCBRS in the conclusions.

To answer **RQ1**, we will firstly consider a *basic* architecture that performs recommendation using only features extracted by a GNN, i.e. using features that are not content-based. Note that there are already existing works that leverage GNNs to learn expressive features representations for a recommendation domain (He et al., 2020). However, most works about GNN focus on the task of classifying nodes in a graph, and do not make use of other deep architectures other than the GNN itself.

Moreover, in order to answer **RQ2**, we will introduce GNNs in the BasicRS architecture using also properties associated to items. Indeed, we will also investigate the introduction of GNNs in the HybridCBRS architecture (Polignano et al., 2021), i.e. using also content-based features that are extracted using BERT.

Finally, we empirically evaluate both the custom architectures based on GNNs for the tripartite graph (namely Two-Step and Two-Way GNNs), and the tweaks for the features-based hybrid architecture (i.e. based on an attention mechanism and residual connections). Please refer to Section 3.4 for details about the proposed models. So, in the next sections, we will show empirical results that will permit to answer those questions.

For this purpose, we will consider five GNN models: GraphSage (Graph SAmple and aggreGaTE), GCN (Graph Convolutional Networks) and GAT (Graph Attention Network) that are the most popular of the three categories, LightGCN that is a lightweight model and DGCF (Liu et al., 2020) that is the most performing one suitable for our case. However, the other ones are obtained by slight variations in the aggregation and update functions (as already exposed in the previous sections). About item-properties information, we will consider the discussion and the proposed models in Section 3.4.

The implementations are based on Spektral (Grattarola and Alippi, 2020) using Tensorflow 2 and Keras. Specifically, Spektral implements the core-layers of GNNs (e.g. GraphSage, GCN and GAT), and also the base classes and utilities. All the models trained using an NVidia RTX 3060 with 12 GiB of memory.

### 5.1 User-Item Setup and Dataset

For our empirical study we rely on the MOVIELENS-1M dataset[2] containing users and movies. Note that it has been preprocessed in order to include properties about movies given by DBpedia[3]. MOVIELENS-1M is a dataset consisting of ratings of users to movies (i.e. the items). The ratings have been binarized such that 0 denotes a negative rating and 1 denotes a positive rating. Moreover, approximately the 20% of ratings are retained for testing. Table 3 show some useful some statistics about the user-item bipartite graph of the MOVIELENS-1M dataset.

[2]https://grouplens.org/datasets/movielens/1m/

[3]https://www.dbpedia.org/

| MOVIELENS-1M DATASET STATISTICS | | | | |
|---|---|---|---|---|
| **User-Item Graph** | | **Item-Properties Graph** | | |
| | | **Relation Setting** | (1) | (2) |
| # Users | 6'036 | # Relations | 7 | 11 |
| # Items | 3'192 | # Properties | 14'615 | 17'554 |
| Users Pos. Avg. Degree | 89.8 | Items Avg. Degree | 21.8 | 24.9 |
| Items Pos. Avg. Degree | 174.3 | Properties Avg. Degree | 4.2 | 4.0 |
| User-Item Sparsity | 95.09% | Item-Properties Sparsity | 99.86% | 99.85% |
| # Total Ratings | 946'772 | # Item-Property Links | 61'434 | 70'341 |
| # Pos. Ratings | 541'788 | | | |
| # Neg. Ratings | 404'984 | | | |

Table 3: MOVIELENS-1M dataset statistics (using the union of training and test sets). On the left side there are statistics about the user-item bipartite graph, while on the right side there are statistics about the item-properties bipartite graph. Note that the latter statistics depend on the relation setting in question. Moreover, for *Avg. Degree* we intend the average number of incoming/outgoing arcs of an entity.

Most works on Graph Neural Networks (GNNs) treat heterogeneous graphs as homogeneous graphs (Wu et al., 2020). So, in our case, we decided to treat the user-item bipartite graph as an undirected homogeneous graph, i.e. considering users and items as entities in our graph. The homogeneous graph is then built as follows. Let $u \in \mathcal{U}$ and $i \in \mathcal{I}$ be an user and an item. We have an undirected arc $(u, i)$ if and only if $u$ expressed a positive rating on the item $i$. Following He et al. (2020), the adjacency matrix of the resulting graph can be written as follows.

$$\mathbf{A}_{ui} = \begin{bmatrix} \mathbf{0} & \mathbf{R}_{ui} \\ \mathbf{R}_{ui}^T & \mathbf{0} \end{bmatrix} \in \{0,1\}^{n_e \times n_e} \tag{22}$$

where $n_e = |\mathcal{U}| + |\mathcal{I}|$, and $\mathbf{R}_{ui} \in \{0,1\}^{|\mathcal{U}| \times |\mathcal{I}|}$. Indeed, the adjacency matrix is a very sparse matrix.

It is important to say that initial features representations of users and entities can be initialized arbitrarily and optimized jointly with the rest of the model. Moreover, multiple GNN layers are stacked sequentially and return updated features representations of both users and items. Finally, the updated features representations of users and items are then passed through a deep feed-forward neural network that computes relevance scores between 0 and 1.

## 5.2 User-Item-Properties Setup

In the user-item-properties setup, we make use of an additional graph related to the MOVIELENS-1M dataset. It is an item-properties bipartite graph which gives to each movie a set of properties such as genre, director, writer, language, and so on and so forth. Thus, in addition to the user-item adjacency matrix $\mathbf{A}_{ui}$ showed in Equation 22, we have another adjacency matrix related to item-properties interactions:

$$\mathbf{A}_{ip} = \begin{bmatrix} \mathbf{0} & \mathbf{R}_{ip} \\ \mathbf{R}_{ip}^T & \mathbf{0} \end{bmatrix} \in \{0,1\}^{n_k \times n_k} \tag{23}$$

where $n_k = |\mathcal{I}| + |\mathcal{P}|$, $\mathcal{P}$ is the set of properties, and $\mathbf{R}_{ip} \in \{0,1\}^{|\mathcal{I}| \times |\mathcal{P}|}$. Eventually, we can also consider the user-item-properties adjacency graph $\mathbf{A}_{uip}$, defined as follows.

$$\mathbf{A}_{uip} = \begin{bmatrix} \mathbf{0} & \mathbf{R}_{ui} & \mathbf{0} \\ \mathbf{R}_{ui}^T & \mathbf{0} & \mathbf{R}_{ip} \\ \mathbf{0} & \mathbf{R}_{ip}^T & \mathbf{0} \end{bmatrix} \in \{0,1\}^{n_e \times n_e} \tag{24}$$

where $n_e = |\mathcal{U}| + |\mathcal{I}| + |\mathcal{P}|$.

Particularly for MOVIELENS-1M, the properties $\mathcal{P}$ have been chosen by considering two sets of relation types settings (RS) with movies:

**RS (1)** $\{subject, director, starring, writer, language, editing, narrator\}$

**RS (2)** $\{subject, director, starring, writer, language, editing, cinematography, musicComposer, country, producer, basedOn\}$

The second set of relation types do not introduce much more entities in the graph, with respect to the first one. However, we decided to experiment with both the relation types settings. Table 3 show some useful some statistics about the item-properties bipartite graphs of the MOVIELENS-1M dataset.

Experiments made in this setup contemplate the usage of the five chosen GNNs models directly on $\mathbf{A}_{uip}$ and, as mentioned in Section 3.4, we also developed two architectures specifically for this setting, i.e. Two-Step GNN and Two-Way GNN.

## 5.3 Basic Architecture Experiments

We compare the results obtained with a version of the BasicRS model that is based on users and items features extracted from Graph Neural Networks (GNNs), as exposed in Section 4.1. Moreover, in order to make the setting harder and effectively assess the expressiveness of GNNs for recommendation, we will consider a very small fraction of the original features dimensionality, hence a fraction of the number of parameters of the entire architecture as well.

Note that we consider as baselines the recommendation performances of BasicRS with users and items features pre-computed using Knowledge Graph Embeddings (KGE) models (e.g. TransD, DistMult) using the user-item bipartite graph.

Finally, for BasicRS-TransD and BasicRS-DistMult we just report the results obtained by V. Digeno using 768-dimensional embeddings for both users and items. Moreover, note that the annotated number of parameters refers to the trainable parameters only. BasicRS with KGE models have additional non-trainable parameters, i.e. the users and items embeddings, which number is not reported in the tables.

### 5.3.1 Hyper-Parameters Selection

For BasicRS models using GCN, GAT and GraphSage we consider, for both users and items an embedding dimension in $\{24, 32, 48, 64, 96, 128\}$. This set is obtained from the combination of two parameters: the number of channels that is chosen in $\{16, 32, 64\}$; and the number of layers that can be 2 or 3 plus the initial embedding which is initialized randomly. Thus, the embeddings are obtained by the concatenation of the sequential propagation layers. However, for LightGCN and DGCF the resulting embedding size are only 16, 32, or 64 both users and items, since they are

| Model (BasicRS) | Reduce | | Dense Units | Channels | # Layers | | $L_2$ Reg. |
|---|---|---|---|---|---|---|---|
| DGCF | Average | | $(24, 24)$ | 8 | 2 | | |
| GAT | Concatenate | | $(32, 32)$ | 8 | 3 | | $10^{-5}$ |
| GCN | Concatenate | $\times$ | $(48, 48)$ | 16 | 2 | $\times$ | $10^{-4}$ |
| GraphSage | Concatenate | | $(64, 64)$ | 16 | 3 | | $10^{-3}$ |
| LightGCN | Average | | $(96, 48)$ | 32 | 2 | | |
| | | | $(128, 64)$ | 32 | 3 | | |

Table 4: Grid search hyper-parameters used for BasicRS models based on Graph Neural Networks (GNNs). The Reduce column tells which method has been used for combining the embeddings obtained by each GNN layer. The Dense Units column stands for the number of units in the both the branches of the BasicRS architecture, after the GNN layers. The Channels column tells the output embedding dimension of each GNN layer.

obtained by averaging the outputs of the layers, and not by concatenation. The best results varying the embedding size are shown in table 17.

Moreover, in our experiments on BasicRS models based on GNNs, we use an adapted version of BasicRS consisting of two dense layers. The first one has a number of units equal to $(K + 1) \times H$, where $K$ is the number of layers, and $H$ the number of channels. Therefore, except for DGCF and LightGCN, it will be equal to the full embedding dimension of users and items. The second layer has half of the units of the first if the first has at least 96 units, otherwise it has the same number. The motivation is to limit the number of parameters, hence reducing the case of overfitting.

Finally, the best results on the test set are showed among different levels of $L_2$ regularization $\{10^{-5}, 10^{-4}, 10^{-3}\}$ applied on the weights of GNNs. GNN-based models are trained by gradient descent for 25 epochs using a batch size of 1024 with Adam (Kingma and Ba, 2015), using a learning rate of $10^{-3}$ and with $\alpha$ set to $0.9$.

We use as loss the *binary cross-entropy* between the actual ratings and the predicted relevance scores. However, in the original implementation of DGCF, the authors use the *bayesian personalized ranking loss* (BPRLoss) (Rendle et al., 2012). We've made a set of experiments using also this loss, which tries to maximize the differences between negative items and positive items for each user. In order to do this, for each batch is sampled half batch size of users, and for each one is sample a positive and a negative item. The applied hyper-parameters of the grid search are summarized in Table 4.

### 5.3.2 Discussion of Results

As shown in Table 5, all GNN models outperform KGEs, since BasicRS-TransD and BasicRS-DistMult are the best performing ones from the previous work. Furthermore, it is notable that the number of trainable parameters is up to ten times lower for GNN models. Among the GNNs models the best resulting one is GAT, that is, the most complex since it uses the attention mechanism.

Models that uses a simpler architecture such as LightGCN and DGCF obtain the best results with more layers and a greater number of channels. Therefore, they need more parameters to compete with other models. So, LightGCN should be designed

| **Model (BasicRS)** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **Training Time (s)** | **# Parameters ($\times 10^5$)** |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5772 | – | – | 11.37 |
| DistMult | – | – | – | 0.5737 | – | – | 11.37 |
| DGCF | 32 | 3 | $10^{-4}$ | 0.5814 | 0.6661 | 280 | 0.33 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5830 | 0.6693 | 629 | 1.68 |
| GCN | 16 | 2 | $10^{-4}$ | 0.5823 | 0.6680 | 211 | 1.68 |
| GraphSage | 32 | 2 | $10^{-4}$ | 0.5829 | 0.6689 | 384 | 3.38 |
| LightGCN | 32 | 3 | $10^{-4}$ | 0.5815 | 0.6653 | 199 | 3.33 |

Table 5: Best results for BasicRS models based on Graph Neural Networks (GNNs) using the user-item graph, and translational and semantic matching Knowledge Graph Embeddings (KGE) models.

to be simpler than GCN, but it ends up to have more parameters. However, DGCF achieves the best ratio score/parameters.

In Table 17 in the Appendix, we can see instead the best experiments comparing the number of channels and the number of layers. In most cases, choosing a higher number of channels produced better results. Moreover, stacking more than two GNN layers generally do not permit to achieve better results, presumably because of the *oversmoothing problem* discussed in Section 2.7. That is, 2 layers are generally sufficient to capture all the required information.

The use of the BPRLoss for DGCF has involved poor results, with an F1@5 score of $0.5597$ for the best experiment with 3 layers, an embedding size of 8 and $L_2$ regularization factor set to $10^{-3}$.

### 5.4 Hybrid Architecture Experiments

We compare the results obtained with a version of the feature-based HybridCBRS model that is based on users and items features extracted from Graph Neural Networks (GNNs), as exposed in Section 4.2. Moreover, similarly to the BasicRS experiments, we will consider a very small fraction of the original features dimensionality, hence a fraction of the number of parameters of the entire architecture as well.

Note that we consider as baselines the recommendation performances of HybridCBRS with users and items collaborative features pre-computed using Knowledge Graph Embeddings (KGE) models (e.g. TransD, DistMult) using the user-item bipartite graph, and content-based features pre-computed using BERT embeddings. Moreover, note that the annotated number of parameters refers to the trainable parameters only. HybridCBRS with KGE models have additional non-trainable parameters, i.e. the users and items embeddings, which number is not reported in the tables.

#### 5.4.1 Hyper-Parameters Selection

The input branches for BERT embedding are composed by two dense layers having a different number of units, i.e. $(256, 64)$. The hyperparameters selection follows the ones for BasicRS explained in Section 5.3.1, except for the dense units for the architecture. That is, they are not halved after the first concatenation layer, since this would decrease the influence of the GNNs with respect to the BERT embeddings. Moreover,

| **Model (HybridCBRS)** | **Reduce** | | **Dense Units** | **Channels** | **# Layers** | | **$L_2$ Reg.** |
|---|---|---|---|---|---|---|---|
| DGCF | Average | | $(24, 24)$ | 8 | 2 | | |
| GAT | Concatenate | | $(32, 32)$ | 8 | 3 | | $10^{-5}$ |
| GCN | Concatenate | $\times$ | $(48, 48)$ | 16 | 2 | $\times$ | $10^{-4}$ |
| GraphSage | Concatente | | $(64, 64)$ | 16 | 3 | | $10^{-3}$ |
| LightGCN | Average | | $(96, 96)$ | 32 | 2 | | |
| | | | $(128, 128)$ | 32 | 3 | | |

Table 6: Grid search hyper-parameters used for HybridCBRS models based on Graph Neural Networks (GNNs). The Reduce column tells which method has been used for combining the embeddings obtained by each GNN layer. The Dense Units column stands for the number of units in the HybridCBRS architecture for GNN embeddings branches. The Channels column tells the output embedding dimension of each GNN layer.

the number of units in the rest of the networks are chosen accordingly to the dense units of the initial branches of the architecture. The applied hyperparameters of the grid search are summarized in Table 6.

Furthermore, experiments with BPRLoss related to DGCF have been discarded, since for the BasicRS architecture we obtained a poor F1@5 score with respect to the other settings. The best results related to the embedding size and the number of GNN layers are shown in Table 18 in the Appendix.

Finally, we perform experiments regarding the feature-based HybridCBRS architecture with the two tweaks described in Section 4.3. That is, we made experiments with the classic feature-based HybridCBRS but with the last concatenation step substituted by an attention mechanism and a dense network with residual connections. To do so, we fix the number of GNN layers to be 2 and the $L_2$ regularization factor to $10^{-4}$, since we noticed that in general almost all of the models with GNNs until now achieved the best results with such hyperparameters. Moreover, we show the results with different GNN models and different number of channels.

### 5.4.2 Discussion of Results

Even if with a little difference, as shown in Table 7, GAT turns out to be the best model also with the *hybrid* architecture. The discussion about the simpler models is confirmed here, as DGCF and LightGCN performs better with greater layer sizes.

GNN models having an higher number of trainable parameters (i.e. GCN, GraphSage and GAT) achieve better results even if using a small number of channels. Moreover, in the same way to what found for the BasicRS model, stacking more than two GNN layers generally do not permit to achieve better results.

Finally, Table 8 shows the results given by the implementation of our tweaks for the feature-based HybridCBRS architecture. As one can see, the original architecture based on concatenation-only as features combination still achieves the best results (alternatively, the others' improvements are negligible) in 6/15 experiments. So, the introduction of a more complex architecture based on either an attention mechanism or residual connections is competitive in some cases in terms of performances. Please refer to Table 19 in the Appendix for the results for a user-item-properties setting.

| **Model (HybridCBRS)** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **Training Time (s)** | **# Parameters ($\times 10^5$)** |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5805 | – | – | 8.80 |
| DistMult | – | – | – | 0.5800 | – | – | 8.80 |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5842 | 0.6685 | 216 | 5.53 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5845 | 0.6703 | 483 | 6.20 |
| GCN | 16 | 2 | $10^{-4}$ | 0.5849 | 0.6704 | 214 | 6.19 |
| GraphSage | 32 | 2 | $10^{-4}$ | 0.5851 | 0.6702 | 446 | 8.05 |
| LightGCN | 16 | 2 | $10^{-4}$ | 0.5838 | 0.6691 | 210 | 6.16 |

Table 7: Best results for HybridCBRS models based on Graph Neural Networks (GNNs) using the user-item graph, and translational and semantic matching Knowledge Graph Embeddings (KGE) models.

| HybridCBRS — User-Item Graph | | | | |
|---|---|---|---|---|
| **Model** | **Channels** | **F1@5 Original** | **F1@5 w/ Attention** | **F1@5 w/ Residual** |
| DGCF | 8 | **0.5842** | 0.5834 | 0.5811 |
| | 16 | 0.5827 | 0.5825 | **0.5837** |
| | 32 | **0.5837** | 0.5833 | 0.5827 |
| GAT | 8 | 0.5825 | **0.5829** | 0.5824 |
| | 16 | 0.5845 | **0.5850** | 0.5818 |
| | 32 | **0.5835** | 0.5833 | 0.5826 |
| GCN | 8 | 0.5824 | 0.5834 | **0.5842** |
| | 16 | **0.5849** | 0.5841 | 0.5834 |
| | 32 | 0.5823 | **0.5852** | 0.5829 |
| GraphSage | 8 | 0.5834 | 0.5835 | **0.5847** |
| | 16 | 0.5821 | **0.5828** | 0.5823 |
| | 32 | **0.5851** | 0.5822 | 0.5845 |
| LightGCN | 8 | 0.5825 | 0.5820 | **0.5835** |
| | 16 | **0.5838** | 0.5827 | 0.5820 |
| | 32 | 0.5826 | 0.5825 | **0.5832** |

Table 8: Comparison of results of the original architecture (i.e. concatenation only) with respect to the introduction of an attention mechanism and residual connections in the feature-based HybridCBRS architecture. Best results with respect to the original architecture are showed in bold.

| Model (BasicRS) | Channels | # Layers | $L_2$ Reg. | F1@5 | F1@10 | Training Time (s) | # Parameters ($\times 10^5$) |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5824 | – | – | 11.37 |
| DistMult | – | – | – | 0.5790 | – | – | 11.37 |
| DGCF | 32 | 2 | $10^{-4}$ | 0.5820 | 0.6672 | 247 | 8.37 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5828 | 0.6680 | 538 | 4.02 |
| GCN | 16 | 3 | $10^{-4}$ | 0.5831 | 0.6690 | 211 | 4.09 |
| GraphSage | 8 | 2 | $10^{-4}$ | 0.5837 | 0.6692 | 338 | 2.01 |
| LightGCN | 16 | 2 | $10^{-4}$ | 0.5814 | 0.6656 | 167 | 3.98 |

Table 9: Best results for BasicRS models based on Graph Neural Networks (GNNs) using the user-item-properties graph with relation setting (1), and translational and semantic matching Knowledge Graph Embeddings (KGE) models.

| Model (BasicRS) | Channels | # Layers | $L_2$ Reg. | F1@5 | F1@10 | Training Time (s) | # Parameters ($\times 10^5$) |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5824 | – | – | 11.37 |
| DistMult | – | – | – | 0.5790 | – | – | 11.37 |
| DGCF | 8 | 3 | $10^{-4}$ | 0.5813 | 0.6655 | 143 | 3.06 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5829 | 0.6680 | 516 | 4.49 |
| GCN | 16 | 2 | $10^{-4}$ | 0.5837 | 0.6692 | 130 | 4.49 |
| GraphSage | 16 | 2 | $10^{-4}$ | 0.5835 | 0.6694 | 320 | 4.49 |
| LightGCN | 16 | 2 | $10^{-4}$ | 0.5831 | 0.6683 | 133 | 4.45 |

Table 10: Best results for BasicRS models based on Graph Neural Networks (GNNs) using the user-item-properties graph with relation setting (2), and translational and semantic matching Knowledge Graph Embeddings (KGE) models.

## 5.5 Basic Architecture Experiments (with Properties)

Starting from the user-item-properties graph, we made experiments using the BasicRS architecture, with Graph Neural Networks (GNNs) applied directly using the complete tripartite graph. Note that the same hyperparameters grid search showed in Table 4 has been followed and applied. The best results for relation settings (1) and (2) for each model are respectively summarized in Table 9 and Table 10. However, we redirect to Table 15 in the conclusions for a full overview of the best models' configurations.

Finally, for BasicRS-TransD and BasicRS-DistMult we will just report the results obtained by V. Digeno using 768-dimensional embeddings for both users and items, which are extracted by leveraging the entire user-item-properties graph with relation types setting (2).

We also experimented with less naive approaches, i.e. using the BasicRS architecture with Two-Step and Two-Way GNN models described in Section 3.4. In this way, we also empirically assess the application of GNN models where users and properties are treated differently. However, these are trained with only the relation types setting (1), since we preferred to stop the experimenting this these models for reasons to be discussed later.

| Model (BasicRS) Two-Step | Channels | # Layers | $L_2$ Reg. | F1@5 | F1@10 | Training Time (s) | # Parameters ($\times 10^5$) |
|---|---|---|---|---|---|---|---|
| DGCF | 8 | 2 | $10^{-4}$ | 0.5807 | 0.6669 | 146 | 2.54 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5785 | 0.6642 | 404 | 2.01 |
| GCN | 16 | 2 | $10^{-4}$ | 0.5785 | 0.6666 | 143 | 4.02 |
| GraphSage | 8 | 3 | $10^{-3}$ | 0.5801 | 0.6665 | 311 | 2.04 |
| LightGCN | 16 | 2 | $10^{-3}$ | 0.5809 | 0.6668 | 278 | 4.04 |

Table 11: Best results for every Graph Neural Networks (GNN) in the BasicRS architecture with Two-Step GNN models, using the user-item-properties graph with relation types setting (1).

| Model (BasicRS) Two-Way | Channels | # Layers | $L_2$ Reg. | F1@5 | F1@10 | Training Time (s) | # Parameters ($\times 10^5$) |
|---|---|---|---|---|---|---|---|
| DGCF | 16 | 3 | $10^{-4}$ | 0.5820 | 0.6662 | 996 | 7.81 |
| GAT | 16 | 3 | $10^{-4}$ | 0.5815 | 0.6657 | 4049 | 6.44 |
| GCN | 16 | 3 | $10^{-4}$ | 0.5805 | 0.6662 | 684 | 6.44 |
| GraphSage | 16 | 2 | $10^{-4}$ | 0.5815 | 0.6681 | 2183 | 6.38 |
| LightGCN | 16 | 2 | $10^{-4}$ | 0.5793 | 0.6647 | 527 | 6.32 |

Table 12: Best results for every Graph Neural Networks (GNN) in the BasicRS architecture with Two-Way GNN models, using the user-item-properties graph with relation types setting (1).

Note that, specifically for the Two-Way GNN models, due to the computational burden added by the user-properties adjacency matrix (which is relatively less sparse compared to the user-item and the utem-properties adjacency matrix), the settings with an embedding size of 32 cannot be applied because that would involve larger weight matrices, hence causing an out-of-memory error. Furthermore, the $L_2$ regularizer of $10^{-5}$ has been removed from the grid search since it performed worse in most of previous experiments, and also because Two-Way GNN models require a lot of training time. Finally, the best results for the Two-Step and Two-Way GNN models are summarized respectively in Tables 11 and 12.

Except for LightGCN and GAT, the other models benefits from the insertion of properties information, in terms of F1@5 scores. However, we cannot determine if the additional relations in setting (2) are useful, because some models perform better, and others not, thus, the formers have received that additional information as noise decreasing the fitness of the model. The newly introduced architectures Two-Step and Two-Way obtained similar or even lower performances with respect to the BasicRS models with just the user-item graph, using relation setting (1), as showed in Table 9.

Therefore, the underlying methodology of infusing knowledge into the embeddings in an antecedent step resulted ineffective, but also, especially for Two-Way, inefficient. Those models require up to double of the parameters, since it uses the user-properties adjacency matrix, and can take training times up to 30 minutes with our setup.

| **Model (HybridCBRS)** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **Training Time (s)** | **# Parameters ($\times 10^5$)** |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5842 | – | – | 8.80 |
| DistMult | – | – | – | 0.5793 | – | – | 8.80 |
| DGCF | 16 | 2 | $10^{-4}$ | 0.5837 | 0.6688 | 393 | 8.97 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5846 | 0.6699 | 606 | 8.53 |
| GCN | 16 | 3 | $10^{-4}$ | 0.5851 | 0.6702 | 332 | 8.63 |
| GraphSage | 8 | 2 | $10^{-4}$ | 0.5847 | 0.6702 | 388 | 6.52 |
| LightGCN | 16 | 2 | $10^{-4}$ | 0.5850 | 0.6694 | 266 | 8.50 |

Table 13: Best results for HybridCBRS models based on Graph Neural Networks (GNNs) using the user-item-properties graph with relation setting (1), and translational and semantic matching Knowledge Graph Embeddings (KGE).

| **Model (HybridCBRS)** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **Training Time (s)** | **# Parameters ($\times 10^5$)** |
|---|---|---|---|---|---|---|---|
| TransD | – | – | – | 0.5842 | – | – | 8.80 |
| DistMult | – | – | – | 0.5793 | – | – | 8.80 |
| DGCF | 8 | 3 | $10^{-5}$ | 0.5831 | 0.6679 | 231 | 7.57 |
| GAT | 16 | 2 | $10^{-4}$ | 0.5845 | 0.6689 | 556 | 9.00 |
| GCN | 16 | 3 | $10^{-4}$ | 0.5842 | 0.6685 | 242 | 9.10 |
| GraphSage | 32 | 2 | $10^{-4}$ | 0.5859 | 0.6700 | 524 | 13.66 |
| LightGCN | 16 | 3 | $10^{-4}$ | 0.5841 | 0.6687 | 231 | 9.03 |

Table 14: Best results for HybridCBRS models based on Graph Neural Networks (GNNs) using the user-item-properties graph with relation setting (2), and translational and semantic matching Knowledge Graph Embeddings (KGE).

## 5.6 Hybrid Architecture Experiments (with Properties)

Due to poor results obtained, the Two-Way and Two-Step models have not been considered for the hybrid setting, i.e. aside with the HybridCBRS architecture using the user-item-properties graph. However, we still investigate the introduction of the user-item-properties graph for Graph Neural Networks (GNNs) in the hybrid setting. As we already said, this is done by simply applying GNN models to the user-item-properties tripartite graph.

The usual grid search for the HybridCBRS architecture showed in Table 6 has been performed. The best results can be seen in Table 13 and 14 respectively for relation settings (1) and (2).

If we compare the obtained results with the results of BasicRS with knowledge in Table 9, and Table 10 we can see an improvement in the performances for every model. However, if we compare them with HybridCBRS without knowledge, as in Table 7 (or see 16 for faster comparison), this is not always true if we consider singularly the relation settings. Thus, the addition of BERT embeddings, always improves the performances; instead the addition of properties could improve the metrics, but not in all cases. However, overall, considering both relation settings, all models except DGCF receive an improvement. This is probably due to the fact that item-properties sub-graph is very sparse, according to the dataset's statistics showed in Table 3. This requires investigation which is out of the scope of this work.

| **Model** | | **F1@5** | | **F1@5** | |
|---|---|---|---|---|---|
| **(BasicRS)** | **UI** | **UIP (1)** | **UIP (2)** | **Two-Step †** | **Two-Way †** |
| TransD | 0.5772 | – | 0.5824 | – | – |
| DistMult | 0.5737 | – | 0.5790 | – | – |
| DGCF | 0.5814 | 0.5820 | 0.5813 | 0.5807 | 0.5820 |
| GAT | 0.5830 | 0.5828 | 0.5829 | 0.5785 | 0.5815 |
| GCN | 0.5823 | 0.5831 | 0.5837 | 0.5785 | 0.5805 |
| GraphSage | 0.5829 | 0.5837 | 0.5835 | 0.5801 | 0.5815 |
| LightGCN | 0.5815 | 0.5814 | 0.5831 | 0.5809 | 0.5793 |

Table 15: Summary table of the best results for the BasicRS model with the user-item graph (UI), and the user-item-properties graph (UIP) with relation settings (1) and (2). Moreover, best results obtained using the Two-Step and Two-Way architectures have been reported. Baselines using TransD and DistMult Knowledge Graph Embeddings (KGE) models are also reported.
† The Two-Step and Two-Way architectures' results refer to the user-item-properties graph with relation setting (1).

## 6 Conclusion

In this work, we showed how Graph Neural Networks (GNNs) are useful for extracting high-quality latent features for users and items in a collaborative-filtering setting. Moreover, the well-known expressiveness of the produced embeddings is again validated by the fact that we used a way lower dimensional space, hence drastically reducing the number of parameters in our models, but still achieving higher performances in terms of precision and recall. The end-to-end training procedure of GNNs is also way simpler to having a two-step learning deep recommender system, where we first learn latent features representations of users and items using Knowledge Graph Embeddings (KGE) models, and then train a deep neural network for relevance classification.

The comparison of performance metrics with the previous works for the *basic* architectures considering only collaborative filtering features (i.e. by using KGE and GNN models) is summarized in Table 15. In the same table, we also show the metrics regarding the user-item-properties tripartite graph setting with two different relation types settings. Moreover, results obtained considering newly introduced custom architectures for the tripartite graph, namely Two-Step and Two-Way GNNs, are also included.

As one can see, we obtained noticeable improvements in terms of F1@5 for the user-item bipartite graph setting, even with a way lower number of parameters as showed in the previous section. However, for the user-item-properties tripartite graph setting we obtained better results only in some cases, while the custom architectures did not achieve better results (with respect to considering only the user-item bipartite graph setting).

The comparison of performance metrics with the previous works for the *hybrid* architectures considering both collaborative features and content-based features (i.e. by using pre-computed BERT embeddings) is summarized in Table 16. As done for the results obtained by the *basic* architecture, we also show the metrics regarding the

| Model (HybridCBRS) | F1@5 | | | F1@5 | | | |
|---|---|---|---|---|---|---|---|
| | | | | **w/ Attention †** | | **w/ Residual †** | |
| | **UI** | **UIP (1)** | **UIP (2)** | **UI** | **UIP (2)** | **UI** | **UIP (2)** |
| TransD | 0.5805 | – | 0.5842 | – | – | – | – |
| DistMult | 0.5800 | – | 0.5793 | – | – | – | – |
| DGCF | 0.5842 | 0.5837 | 0.5831 | 0.5834 | 0.5840 | 0.5837 | 0.5844 |
| GAT | 0.5845 | 0.5846 | 0.5845 | 0.5850 | 0.5846 | 0.5826 | 0.5838 |
| GCN | 0.5849 | 0.5851 | 0.5842 | 0.5852 | 0.5849 | 0.5842 | 0.5834 |
| GraphSage | 0.5851 | 0.5847 | 0.5859 | 0.5835 | 0.5856 | 0.5847 | 0.5836 |
| LightGCN | 0.5838 | 0.5850 | 0.5841 | 0.5827 | 0.5838 | 0.5835 | 0.5833 |

Table 16: Summary table of the best results for the HybridCBRS model with the user-item graph (UI), and the user-item-properties graph (UIP) with relation settings (1) and (2). Moreover, best results obtained using the feature-based hybrid architectures tweaks have been reported. Baselines using TransD and DistMult Knowledge Graph Embeddings (KGE) models are also reported. † The feature-based hybrid architectures tweaks' results refer to the user-item graph (UI), and user-item-properties graph with relation setting (2).

user-item-properties tripartite graph setting with two different relation types settings. Moreover, results obtained by considering the newly introduced feature-based *hybrid* architectures' tweaks are also reported.

We can notice how the use of GNNs improves the models' performances in terms of F1@5 scores also in the setting using both collaborative and content-based features. However, for the user-item-properties tripartite graph setting, we obtained better results only in some cases. The newly introduced tweaks for the feature-based hybrid architecture achieve performances which are quite similar to the original architecture. However, the fact that we achieved better scores in some cases combined with the well-known theoretical benefits of using attention mechanisms and residual connections induce us to think these would require further investigation.

## 6.1 Future Works

The fact that we achieved similar (or even worse) results in the user-item-properties tripartite graph setting made us think about the possibility to enhance the number of relation types and properties in the underlying knowledge graph, since the current ones are too small and too sparse. Therefore, we believe that the inclusion of a richer knowledge graph structure for items would have a good impact on recommendation performances, even with the usage of GNNs as already noted in some previous works briefly described in Section 2, and later discussed in Section 3.

Finally, consider the *hybrid* architecture using GNNs we introduced in this work. Such model still requires the pre-computation of items' and users' latent features based on word embeddings extracted from the textual description (e.g. the plot for movies) of the items. A future work might consider the full integration of a transformer-based language model (e.g. BERT or ERNIE) for the extraction of embeddings for the content-based branch. Such models would be instantiated as pre-trained and fine-tuned with the rest of the *hybrid* architecture in a fully end-to-end fashion.

## References

D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445. AAAI Press, 2020a.

L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *AAAI*, pages 27–34. AAAI Press, 2020b.

D. Grattarola and C. Alippi. Graph neural networks in TensorFlow and Keras with Spektral. *CoRR*, abs/2006.12138, 2020.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, pages 639–648. ACM, 2020.

B. Jin, C. Gao, X. He, D. Jin, and Y. Li. Multi-behavior recommendation with graph convolutional networks. In *SIGIR*, pages 659–668. ACM, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017.

Z. Liu, L. Meng, J. Zhang, and P. S. Yu. Deoscillated graph collaborative filtering. *CoRR*, abs/2011.02100, 2020.

M. Polignano, C. Musto, M. de Gemmis, P. Lops, and G. Semeraro. Together is better: Hybrid recommendations combining graph embeddings and contextualized word representations. In *RecSys*, pages 187–198. ACM, 2021.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012. URL http://arxiv.org/abs/1205.2618.

C. Su, M. Chen, and X. Xie. Graph convolutional matrix completion via relation reconstruction. In *ICSCA*, pages 51–56. ACM, 2021.

P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

H. Wang, F. Zhang, M. Zhang, J. Leskovec, M. Zhao, W. Li, and Z. Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *KDD*, pages 968–977. ACM, 2019a.

H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo. Knowledge graph convolutional networks for recommender systems. In *WWW*, pages 3307–3313. ACM, 2019b.

X. Wang, X. He, Y. Cao, M. Liu, and T. Chua. KGAT: knowledge graph attention network for recommendation. In *KDD*, pages 950–958. ACM, 2019c.

X. Wang, X. He, M. Wang, F. Feng, and T. Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174. ACM, 2019d.

X. Wang, H. Ji, C. Shi, B. Wang, P. Cui, P. S. Yu, and Y. Ye. Heterogeneous graph attention network. *CoRR*, abs/1903.07293, 2019e.

X. Wang, R. Wang, C. Shi, G. Song, and Q. Li. Multi-component graph convolutional collaborative filtering. In *AAAI*, pages 6267–6274. AAAI Press, 2020.

S. Wu, W. Zhang, F. Sun, and B. Cui. Graph neural networks in recommender systems: A survey. *CoRR*, abs/2011.02260, 2020.

R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983. ACM, 2018.

M. Zhang and Y. Chen. Inductive matrix completion based on graph neural networks. In *ICLR*. OpenReview.net, 2020.

J. Zhao, Z. Zhou, Z. Guan, W. Zhao, W. Ning, G. Qiu, and X. He. Intentgc: A scalable graph convolution framework fusing heterogeneous information for recommendation. In *KDD*, pages 2347–2357. ACM, 2019.

L. Zheng, C. Lu, F. Jiang, J. Zhang, and P. S. Yu. Spectral collaborative filtering. In *RecSys*, pages 311–319. ACM, 2018.

## Appendix A Full Experiments Results

| BasicRS — User-Item Graph | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5803 | 0.6643 | 1.01 |
| | 16 | 2 | $10^{-4}$ | 0.5795 | 0.6674 | 1.83 |
| | 32 | 2 | $10^{-4}$ | 0.5814 | 0.6661 | 3.40 |
| | 8 | 3 | $10^{-5}$ | 0.5803 | 0.6669 | 1.10 |
| | 16 | 3 | $10^{-4}$ | 0.5808 | 0.6677 | 1.98 |
| | **32** | **3** | $\mathbf{10^{-4}}$ | **0.5814** | **0.6678** | **3.60** |
| GAT | 8 | 2 | $10^{-5}$ | 0.5790 | 0.6669 | 0.81 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5830** | **0.6693** | **1.69** |
| | 32 | 2 | $10^{-4}$ | 0.5824 | 0.6679 | 3.36 |
| | 8 | 3 | $10^{-4}$ | 0.5812 | 0.6673 | 0.87 |
| | 16 | 3 | $10^{-4}$ | 0.5806 | 0.6667 | 1.78 |
| | 32 | 3 | $10^{-4}$ | 0.5821 | 0.6679 | 3.60 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5814 | 0.6649 | 0.81 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5823** | **0.6680** | **1.68** |
| | 32 | 2 | $10^{-4}$ | 0.5805 | 0.6668 | 3.36 |
| | 8 | 3 | $10^{-4}$ | 0.5809 | 0.6678 | 0.87 |
| | 16 | 3 | $10^{-4}$ | 0.5812 | 0.6689 | 1.78 |
| | 32 | 3 | $10^{-4}$ | 0.5818 | 0.6682 | 3.60 |
| GraphSage | 8 | 2 | $10^{-3}$ | 0.5825 | 0.6682 | 0.81 |
| | 16 | 2 | $10^{-3}$ | 0.5806 | 0.6682 | 1.69 |
| | **32** | **2** | $\mathbf{10^{-4}}$ | **0.5829** | **0.6689** | **3.38** |
| | 8 | 3 | $10^{-3}$ | 0.5813 | 0.6679 | 0.87 |
| | 16 | 3 | $10^{-4}$ | 0.5820 | 0.6710 | 1.78 |
| | 32 | 3 | $10^{-4}$ | 0.5822 | 0.6684 | 3.64 |
| LightGCN | 8 | 2 | $10^{-4}$ | 0.5793 | 0.6658 | 0.80 |
| | 16 | 2 | $10^{-4}$ | 0.5811 | 0.6652 | 1.64 |
| | 32 | 2 | $10^{-4}$ | 0.5808 | 0.6660 | 3.21 |
| | 8 | 3 | $10^{-4}$ | 0.5811 | 0.6667 | 0.85 |
| | 16 | 3 | $10^{-5}$ | 0.5810 | 0.6671 | 1.71 |
| | **32** | **3** | $\mathbf{10^{-4}}$ | **0.5815** | **0.6653** | **3.33** |

Table 17: Comparison of different number of channels and number of layers of the BasicRS architecture with Graph Neural Networks (GNNs), using the user-item graph. Note that the best scores with respect to the $L_2$ regularization factors are reported.

| HybridCBRS — User-Item Graph | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | **8** | **2** | $\mathbf{10^{-4}}$ | **0.5842** | **0.6685** | **5.53** |
| | 16 | 2 | $10^{-3}$ | 0.5829 | 0.6681 | 6.34 |
| | 32 | 2 | $10^{-4}$ | 0.5837 | 0.6695 | 8.07 |
| | 8 | 3 | $10^{-4}$ | 0.5839 | 0.6684 | 5.64 |
| | 16 | 3 | $10^{-4}$ | 0.5840 | 0.6690 | 6.50 |
| | 32 | 3 | $10^{-4}$ | 0.5830 | 0.6696 | 8.36 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5825 | 0.6674 | 5.35 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5845** | **0.6703** | **6.20** |
| | 32 | 2 | $10^{-4}$ | 0.5835 | 0.6699 | 8.03 |
| | 8 | 3 | $10^{-4}$ | 0.5834 | 0.6698 | 5.38 |
| | 16 | 3 | $10^{-4}$ | 0.5838 | 0.6683 | 6.29 |
| | 32 | 3 | $10^{-4}$ | 0.5837 | 0.6678 | 8.37 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5824 | 0.6692 | 5.35 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5849** | **0.6704** | **6.19** |
| | 32 | 2 | $10^{-4}$ | 0.5823 | 0.6695 | 8.03 |
| | 8 | 3 | $10^{-4}$ | 0.5848 | 0.6698 | 5.38 |
| | 16 | 3 | $10^{-4}$ | 0.5831 | 0.6697 | 6.29 |
| | 32 | 3 | $10^{-4}$ | 0.5832 | 0.6699 | 8.37 |
| GraphSage | 8 | 2 | $10^{-3}$ | 0.5846 | 0.6691 | 5.35 |
| | 16 | 2 | $10^{-3}$ | 0.5840 | 0.6692 | 6.20 |
| | **32** | **2** | $\mathbf{10^{-4}}$ | **0.5851** | **0.6702** | **8.05** |
| | 8 | 3 | $10^{-4}$ | 0.5849 | 0.6696 | 5.38 |
| | 16 | 3 | $10^{-3}$ | 0.5835 | 0.6694 | 6.30 |
| | 32 | 3 | $10^{-3}$ | 0.5828 | 0.6684 | 8.40 |
| LightGCN | 8 | 2 | $10^{-4}$ | 0.5825 | 0.6677 | 5.34 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5838** | **0.6691** | **6.16** |
| | 32 | 2 | $10^{-4}$ | 0.5826 | 0.6692 | 7.88 |
| | 8 | 3 | $10^{-4}$ | 0.5822 | 0.6681 | 5.36 |
| | 16 | 3 | $10^{-5}$ | 0.5826 | 0.6689 | 6.22 |
| | 32 | 3 | $10^{-4}$ | 0.5833 | 0.6686 | 8.09 |

Table 18: Comparison of different number of channels and number of layers of the HybridCBRS architecture with Graph Neural Networks (GNNs), using the user-item graph. Note that the best scores with respect to the $L_2$ regularization factors are reported.

| HybridCBRS — User-Item-Properties Graph (Relation Setting 2) | | | | |
|---|---|---|---|---|
| **Model** | **Channels** | **F1@5 Original** | **F1@5 w/ Attention** | **F1@5 w/ Residual** |
| DGCF | 8 | 0.5828 | 0.5814 | **0.5831** |
| | 16 | 0.5824 | 0.5828 | **0.5844** |
| | 32 | 0.5820 | **0.5840** | 0.5826 |
| GAT | 8 | **0.5839** | 0.5815 | 0.5826 |
| | 16 | **0.5845** | 0.5819 | 0.5838 |
| | 32 | 0.5833 | **0.5846** | 0.5838 |
| GCN | 8 | 0.5818 | 0.5819 | **0.5834** |
| | 16 | 0.5835 | **0.5849** | 0.5838 |
| | 32 | 0.5818 | **0.5842** | 0.5824 |
| GraphSage | 8 | 0.5830 | **0.5831** | 0.5816 |
| | 16 | 0.5843 | **0.5856** | 0.5830 |
| | 32 | **0.5859** | 0.5830 | 0.5836 |
| LightGCN | 8 | **0.5824** | 0.5806 | 0.5818 |
| | 16 | **0.5841** | 0.5822 | 0.5833 |
| | 32 | 0.5804 | **0.5838** | 0.5833 |

Table 19: Comparison of results of the original architecture (i.e. concatenation only) with respect to the introduction of an attention mechanism and residual connections in the feature-based HybridCBRS architecture, using the user-item-properties graph with relation setting (2). Best results with respect to the original architecture are showed in bold. Note that the showed results refer to the 2 layers setting with a $L_2$ regularization factor of $10^{-4}$.

| BasicRS — User-Item-Properties Graph (Relation setting 1) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **L$_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5801 | 0.6650 | 2.47 |
| | 16 | 2 | $10^{-4}$ | 0.5814 | 0.6670 | 4.46 |
| | **32** | **2** | $\mathbf{10^{-4}}$ | **0.5820** | **0.6672** | **8.37** |
| | 8 | 3 | $10^{-4}$ | 0.5810 | 0.6657 | 2.73 |
| | 16 | 3 | $10^{-4}$ | 0.5816 | 0.6651 | 4.76 |
| | 32 | 3 | $10^{-4}$ | 0.5805 | 0.6638 | 8.72 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5808 | 0.6660 | 2.01 |
| | 16 | 2 | $10^{-4}$ | 0.5828 | 0.6680 | 4.02 |
| | **32** | **2** | $\mathbf{10^{-4}}$ | **0.5824** | **0.6689** | **8.04** |
| | 8 | 3 | $10^{-4}$ | 0.5823 | 0.6674 | 2.04 |
| | 16 | 3 | $10^{-4}$ | 0.5808 | 0.6655 | 4.09 |
| | 32 | 3 | $10^{-4}$ | 0.5828 | 0.6691 | 8.19 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5816 | 0.6683 | 2.01 |
| | 16 | 2 | $10^{-4}$ | 0.5820 | 0.6688 | 4.02 |
| | 32 | 2 | $10^{-4}$ | 0.5799 | 0.6675 | 8.03 |
| | 8 | 3 | $10^{-4}$ | 0.5813 | 0.6668 | 2.04 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5831** | **0.6690** | **4.09** |
| | 32 | 3 | $10^{-4}$ | 0.5800 | 0.6681 | 8.19 |
| GraphSage | **8** | **2** | $\mathbf{10^{-4}}$ | **0.5837** | **0.6692** | **2.01** |
| | 16 | 2 | $10^{-4}$ | 0.5828 | 0.6689 | 4.02 |
| | 32 | 2 | $10^{-3}$ | 0.5819 | 0.6682 | 8.06 |
| | 8 | 3 | $10^{-4}$ | 0.5824 | 0.6685 | 2.04 |
| | 16 | 3 | $10^{-4}$ | 0.5821 | 0.6693 | 4.10 |
| | 32 | 3 | $10^{-4}$ | 0.5822 | 0.6683 | 8.21 |
| LightGCN | 8 | 2 | $10^{-4}$ | 0.5792 | 0.6642 | 2.00 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5814** | **0.6656** | **3.98** |
| | 32 | 2 | $10^{-4}$ | 0.5805 | 0.6654 | 7.89 |
| | 8 | 3 | $10^{-4}$ | 0.5811 | 0.6658 | 2.02 |
| | 16 | 3 | $10^{-4}$ | 0.5799 | 0.6650 | 4.04 |
| | 32 | 3 | $10^{-4}$ | 0.5797 | 0.6646 | 8.00 |

Table 20: Comparison of different number of channels and number of layers of the BasicRS architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (1). Note that the best scores with respect to the $L_2$ regularization factors are reported.

| BasicRS — User-Item-Properties Graph (Relation Setting 2) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5796 | 0.6643 | 2.74 |
| | 16 | 2 | $10^{-4}$ | 0.5810 | 0.6654 | 4.99 |
| | 32 | 2 | $10^{-4}$ | 0.5795 | 0.6677 | 9.37 |
| | **8** | **3** | $\mathbf{10^{-4}}$ | **0.5813** | **0.6655** | **3.06** |
| | 16 | 3 | $10^{-4}$ | 0.5808 | 0.6652 | 5.32 |
| | 32 | 3 | $10^{-4}$ | 0.5805 | 0.6678 | 9.75 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5817 | 0.6667 | 2.22 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5829** | **0.668** | **4.49** |
| | 32 | 2 | $10^{-4}$ | 0.5813 | 0.6672 | 8.98 |
| | 8 | 3 | $10^{-4}$ | 0.5803 | 0.6662 | 2.27 |
| | 16 | 3 | $10^{-4}$ | 0.5808 | 0.6674 | 4.59 |
| | 32 | 3 | $10^{-4}$ | 0.5822 | 0.6684 | 9.22 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5808 | 0.6652 | 2.22 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5837** | **0.6692** | **4.49** |
| | 32 | 2 | $10^{-4}$ | 0.5801 | 0.6683 | 8.98 |
| | 8 | 3 | $10^{-4}$ | 0.5819 | 0.6687 | 2.27 |
| | 16 | 3 | $10^{-4}$ | 0.5796 | 0.6673 | 4.58 |
| | 32 | 3 | $10^{-4}$ | 0.5824 | 0.6684 | 9.22 |
| GraphSage | 8 | 2 | $10^{-4}$ | 0.5827 | 0.6694 | 2.22 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5835** | **0.6694** | **4.49** |
| | 32 | 2 | $10^{-4}$ | 0.5827 | 0.6687 | 9.00 |
| | 8 | 3 | $10^{-4}$ | 0.5817 | 0.6681 | 2.27 |
| | 16 | 3 | $10^{-4}$ | 0.5829 | 0.6687 | 4.59 |
| | 32 | 3 | $10^{-4}$ | 0.5814 | 0.6687 | 9.25 |
| LightGCN | 8 | 2 | $10^{-5}$ | 0.5787 | 0.6644 | 2.21 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5831** | **0.6683** | **4.45** |
| | 32 | 2 | $10^{-4}$ | 0.5805 | 0.6651 | 8.83 |
| | 8 | 3 | $10^{-5}$ | 0.5804 | 0.6660 | 2.25 |
| | 16 | 3 | $10^{-5}$ | 0.5814 | 0.6681 | 4.51 |
| | 32 | 3 | $10^{-5}$ | 0.5805 | 0.6674 | 8.94 |

Table 21: Comparison of different number of channels and number of layers of the BasicRS architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (2). Note that the best scores with respect to the $L_2$ regularization factors are reported.

| BasicRS Two-Step — User-Item-Properties Graph (Relation Setting 1) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | **8** | **2** | $\mathbf{10^{-4}}$ | **0.5807** | **0.6669** | **2.54** |
| | 16 | 2 | $10^{-3}$ | 0.5793 | 0.6663 | 4.59 |
| | 32 | 2 | $10^{-4}$ | 0.5783 | 0.6671 | 8.54 |
| | 8 | 3 | $10^{-3}$ | 0.5802 | 0.6661 | 2.83 |
| | 16 | 3 | $10^{-4}$ | 0.5797 | 0.6669 | 4.86 |
| | 32 | 3 | $10^{-4}$ | 0.5803 | 0.6667 | 8.82 |
| GAT | **8** | **2** | $\mathbf{10^{-4}}$ | **0.5785** | **0.6642** | **2.01** |
| | 16 | 2 | $10^{-4}$ | 0.5779 | 0.6646 | 4.10 |
| | 32 | 2 | $10^{-4}$ | 0.5761 | 0.6630 | 8.21 |
| | 8 | 3 | $10^{-3}$ | 0.5766 | 0.6641 | 2.04 |
| | 16 | 3 | $10^{-5}$ | 0.5774 | 0.6645 | 4.10 |
| | 32 | 3 | $10^{-4}$ | 0.5774 | 0.6646 | 8.21 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5783 | 0.6668 | 3.11 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5785** | **0.6666** | **4.02** |
| | 32 | 2 | $10^{-3}$ | 0.5760 | 0.6653 | 8.21 |
| | 8 | 3 | $10^{-4}$ | 0.5793 | 0.6665 | 2.04 |
| | 16 | 3 | $10^{-4}$ | 0.5777 | 0.6658 | 4.10 |
| | 32 | 3 | $10^{-3}$ | 0.5771 | 0.6651 | 8.21 |
| GraphSage | 8 | 2 | $10^{-3}$ | 0.5795 | 0.6663 | 3.12 |
| | 16 | 2 | $10^{-4}$ | 0.5782 | 0.6646 | 4.11 |
| | 32 | 2 | $10^{-3}$ | 0.5782 | 0.6653 | 8.10 |
| | **8** | **3** | $\mathbf{10^{-3}}$ | **0.5801** | **0.6665** | **2.04** |
| | 16 | 3 | $10^{-3}$ | 0.5792 | 0.6662 | 4.11 |
| | 32 | 3 | $10^{-4}$ | 0.5787 | 0.6671 | 8.25 |
| LightGCN | 8 | 2 | $10^{-4}$ | 0.5788 | 0.6651 | 2.00 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5809** | **0.6668** | **4.04** |
| | 32 | 2 | $10^{-4}$ | 0.5809 | 0.6677 | 7.89 |
| | 8 | 3 | $10^{-4}$ | 0.5770 | 0.6648 | 2.02 |
| | 16 | 3 | $10^{-5}$ | 0.5789 | 0.6649 | 4.04 |
| | 32 | 3 | $10^{-3}$ | 0.5810 | 0.6662 | 8.00 |

Table 22: Comparison of different number of channels and number of layers of the BasicRS Two-Step architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (1). Note that the best scores with respect to the $L_2$ regularization factors are reported.

| BasicRS Two-Way — User-Item-Properties Graph (Relation Setting 1) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5816 | 0.6668 | 4.12 |
| | 16 | 2 | $10^{-4}$ | 0.5801 | 0.6649 | 7.27 |
| | 8 | 3 | $10^{-4}$ | 0.5800 | 0.6653 | 4.62 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5820** | **0.6662** | **7.81** |
| GAT | 8 | 2 | $10^{-4}$ | 0.5783 | 0.6650 | 3.18 |
| | 16 | 2 | $10^{-4}$ | 0.5797 | 0.6648 | 6.37 |
| | 8 | 3 | $10^{-4}$ | 0.5785 | 0.6647 | 3.21 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5815** | **0.6657** | **6.44** |
| GCN | 8 | 2 | $10^{-4}$ | 0.5787 | 0.6654 | 3.18 |
| | 16 | 2 | $10^{-4}$ | 0.5787 | 0.6655 | 6.37 |
| | 8 | 3 | $10^{-4}$ | 0.5781 | 0.6646 | 3.21 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5805** | **0.6662** | **6.44** |
| GraphSage | 8 | 2 | $10^{-4}$ | 0.5806 | 0.6685 | 3.18 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5815** | **0.6681** | **6.38** |
| | 8 | 3 | $10^{-4}$ | 0.5804 | 0.6680 | 3.21 |
| | 16 | 3 | $10^{-4}$ | 0.5793 | 0.6658 | 6.46 |
| LightGCN | 8 | 2 | $10^{-4}$ | 0.5789 | 0.6644 | 3.17 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5793** | **0.6647** | **6.32** |
| | 8 | 3 | $10^{-4}$ | 0.5774 | 0.6643 | 3.19 |
| | 16 | 3 | $10^{-4}$ | 0.5784 | 0.6647 | 6.38 |

Table 23: Comparison of different number of channels and number of layers of the BasicRS Two-Way architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (1). Note that the best scores with respect to the $L_2$ regularization factors are reported.

| HybridCBRS — User-Item-Properties Graph (Relation Setting 1) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **$L_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5836 | 0.6694 | 6.99 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5837** | **0.6688** | **8.97** |
| | 32 | 2 | $10^{-4}$ | 0.5824 | 0.6687 | 13.04 |
| | 8 | 3 | $10^{-3}$ | 0.5804 | 0.6664 | 7.25 |
| | 16 | 3 | $10^{-4}$ | 0.5825 | 0.6692 | 9.27 |
| | 32 | 3 | $10^{-4}$ | 0.5842 | 0.6685 | 13.48 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5826 | 0.6693 | 6.52 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5846** | **0.6699** | **8.53** |
| | 32 | 2 | $10^{-4}$ | 0.5820 | 0.6694 | 12.70 |
| | 8 | 3 | $10^{-4}$ | 0.5840 | 0.6691 | 6.55 |
| | 16 | 3 | $10^{-4}$ | 0.5833 | 0.6693 | 8.63 |
| | 32 | 3 | $10^{-4}$ | 0.5824 | 0.6682 | 13.05 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5837 | 0.6702 | 6.52 |
| | 16 | 2 | $10^{-4}$ | 0.5832 | 0.6698 | 8.53 |
| | 32 | 2 | $10^{-4}$ | 0.5841 | 0.6704 | 12.70 |
| | 8 | 3 | $10^{-4}$ | 0.5833 | 0.6692 | 6.55 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5851** | **0.6702** | **8.63** |
| | 32 | 3 | $10^{-4}$ | 0.5822 | 0.6694 | 13.04 |
| GraphSage | **8** | **2** | $\mathbf{10^{-4}}$ | **0.5847** | **0.6702** | **6.52** |
| | 16 | 2 | $10^{-4}$ | 0.5825 | 0.6683 | 8.54 |
| | 32 | 2 | $10^{-3}$ | 0.5823 | 0.6674 | 12.72 |
| | 8 | 3 | $10^{-4}$ | 0.5846 | 0.6701 | 6.55 |
| | 16 | 3 | $10^{-3}$ | 0.5835 | 0.6686 | 8.64 |
| | 32 | 3 | $10^{-4}$ | 0.5841 | 0.6682 | 13.07 |
| LightGCN | 8 | 2 | $10^{-5}$ | 0.5836 | 0.6682 | 6.51 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5850** | **0.6694** | **8.50** |
| | 32 | 2 | $10^{-5}$ | 0.5830 | 0.6696 | 12.56 |
| | 8 | 3 | $10^{-4}$ | 0.5824 | 0.6681 | 6.53 |
| | 16 | 3 | $10^{-4}$ | 0.5830 | 0.6692 | 8.56 |
| | 32 | 3 | $10^{-5}$ | 0.5812 | 0.6679 | 12.77 |

Table 24: Comparison of different number of channels and number of layers of the HybridRS architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (1). Note that the best scores with respect to the $L_2$ regularization factors are reported.

| HybridCBRS — User-Item-Properties Graph (Relation Setting 2) | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **Channels** | **# Layers** | **L$_2$ Reg.** | **F1@5** | **F1@10** | **# Parameters ($\times 10^5$)** |
| DGCF | 8 | 2 | $10^{-4}$ | 0.5828 | 0.6699 | 7.28 |
| | 16 | 2 | $10^{-4}$ | 0.5824 | 0.6686 | 9.50 |
| | 32 | 2 | $10^{-4}$ | 0.5820 | 0.6688 | 14.04 |
| | **8** | **3** | $\mathbf{10^{-5}}$ | **0.5831** | **0.6679** | **7.57** |
| | 16 | 3 | $10^{-4}$ | 0.5830 | 0.6686 | 9.83 |
| | 32 | 3 | $10^{-4}$ | 0.5816 | 0.6691 | 14.51 |
| GAT | 8 | 2 | $10^{-4}$ | 0.5839 | 0.6699 | 6.76 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5845** | **0.6689** | **9.00** |
| | 32 | 2 | $10^{-4}$ | 0.5833 | 0.6694 | 13.65 |
| | 8 | 3 | $10^{-4}$ | 0.5829 | 0.6698 | 6.79 |
| | 16 | 3 | $10^{-4}$ | 0.5840 | 0.6696 | 9.10 |
| | 32 | 3 | $10^{-4}$ | 0.5841 | 0.6693 | 13.99 |
| GCN | 8 | 2 | $10^{-4}$ | 0.5818 | 0.6699 | 6.76 |
| | 16 | 2 | $10^{-4}$ | 0.5835 | 0.6703 | 9.00 |
| | 32 | 2 | $10^{-4}$ | 0.5818 | 0.6695 | 13.64 |
| | 8 | 3 | $10^{-4}$ | 0.5822 | 0.6690 | 6.79 |
| | **16** | **3** | $\mathbf{10^{-4}}$ | **0.5842** | **0.6685** | **9.10** |
| | 32 | 3 | $10^{-4}$ | 0.5829 | 0.6689 | 13.98 |
| GraphSage | 8 | 2 | $10^{-4}$ | 0.5830 | 0.6690 | 6.76 |
| | 16 | 2 | $10^{-4}$ | 0.5843 | 0.6696 | 9.01 |
| | **32** | **2** | $\mathbf{10^{-4}}$ | **0.5859** | **0.6700** | **13.60** |
| | 8 | 3 | $10^{-4}$ | 0.5840 | 0.6687 | 6.79 |
| | 16 | 3 | $10^{-4}$ | 0.5839 | 0.6701 | 9.11 |
| | 32 | 3 | $10^{-3}$ | 0.5831 | 0.6688 | 14.01 |
| LightGCN | 8 | 2 | $10^{-5}$ | 0.5824 | 0.6686 | 6.75 |
| | **16** | **2** | $\mathbf{10^{-4}}$ | **0.5841** | **0.6689** | **8.97** |
| | 32 | 2 | $10^{-4}$ | 0.5804 | 0.6683 | 13.50 |
| | 8 | 3 | $10^{-5}$ | 0.5830 | 0.6696 | 6.77 |
| | 16 | 3 | $10^{-4}$ | 0.5841 | 0.6687 | 9.03 |
| | 32 | 3 | $10^{-5}$ | 0.5826 | 0.6691 | 13.71 |

Table 25: Comparison of different number of channels and number of layers of the HybridRS architecture with Graph Neural Networks (GNNs), using the user-item-properties graph with relation setting (2). Note that the best scores with respect to the $L_2$ regularization factors are reported.