Multi-Relational Graph based Heterogeneous Multi-Task Learning in Community Question Answering

Zizheng Lin\textsuperscript{1}, Haowen Ke\textsuperscript{1}, Ngo-Yin Wong\textsuperscript{1}, Jiaxin Bai\textsuperscript{1}, Yangqiu Song\textsuperscript{1}, Huan Zhao\textsuperscript{2}, Junpeng Ye\textsuperscript{3}

\textsuperscript{1}Department of Computer Science and Engineering, HKUST, Hong Kong, China
\textsuperscript{2}Paradigm Inc., Beijing, China
\textsuperscript{3}Tencent Technology (SZ) Co., Ltd., Shenzhen, China

\{zlinai,hkeaa,nywongac,jbai,yqsong\}@cse.ust.hk,zhaohuan@4paradigm.com,jayjpye@tencent.com

\section*{ABSTRACT}
Various data mining tasks have been proposed to study Community Question Answering (CQA) platforms like Stack Overflow. The relatedness between some of these tasks provides useful learning signals to each other via Multi-Task Learning (MTL). However, due to the high heterogeneity of these tasks, few existing works manage to jointly solve them in a unified framework. To tackle this challenge, we develop a multi-relational graph based MTL model called Heterogeneous Multi-Task Graph Isomorphism Network (HMTGIN) which efficiently solves heterogeneous CQA tasks. In each training forward pass, HMTGIN embeds the input CQA forum graph by an extension of Graph Isomorphism Network and skip connections. The embeddings are then shared across all task-specific output layers to compute respective losses. Moreover, two cross-task constraints based on the domain knowledge about tasks’ relationships are used to regularize the joint learning. In the evaluation, the embeddings are shared among different task-specific output layers to make corresponding predictions. To the best of our knowledge, HMTGIN is the first MTL model capable of tackling CQA tasks from the aspect of multi-relational graphs. To evaluate HMTGIN’s effectiveness, we build a novel large-scale multi-relational graph CQA dataset with over two million nodes from Stack Overflow. Extensive experiments show that: (1) HMTGIN is superior to all baselines on five tasks; (2) The proposed MTL strategy and cross-task constraints have substantial advantages.

\section*{CCS CONCEPTS}
\textbullet{} Information systems → Data mining.

\section*{KEYWORDS}
Community Question Answering, Heterogeneous Multi-Task Learning, Multi-Relational Graph, Cross-task Constraint

\section*{INTRODUCTION}
Community Question Answering (CQA) forums like Stack Overflow\footnote{https://stackoverflow.com/} help millions of users to seek solutions or share their knowledge. Various CQA tasks like duplicate question detection and answer recommendation have been extensively studied\cite{31}. Due to rich interconnections between components of a CQA platform, some CQA tasks are related to each other. For example, tag recommendation can be solved using a link prediction model and the semantic representation of keywords reflected by this task can also help evaluate a similar link prediction model in duplicate question detection. Moreover, duplicate question detection would also help personalize answer recommendation as they both evaluate text similarities, although the latter should be formulated as a ranking problem. Thus, it is natural to use Multi-Task Learning (MTL)\cite{5} to jointly solve different CQA tasks to obtain better overall performance.

However, most existing MTL frameworks only consider similar tasks\cite{25,48} like considering classification and regression of the same learning features\cite{47}, or assuming that the features are different but label spaces are the same across tasks\cite{16}. Regarding heterogeneous tasks (i.e., tasks with very different properties such as contrasting objective functions and learning features), these MTL frameworks cannot be effective. In the case of tackling CQA tasks, there are two major challenges for heterogeneous MTL (e.g., jointly solve link prediction, classification, and ranking over different types of nodes).

First, it is non-trivial to share features for heterogeneous CQA data in the semantic space. Specifically, different entities, like users and answers barely share similar semantic space, as they are usually connected through complicated paths in different types of relations. It may be not sufficient for the algorithm to learn the feature dependency, which is crucial for effectively sharing features across tasks, directly from training feature representations. Thus, a multi-relational graph based model should be considered to build relationships among entities in a CQA platform.

Second, there are explicit relationships among output labels in the label space. However, to our best knowledge, existing MTL algorithms do not explicitly model the label relations. For example, in CQA, a more reputable user’s answer is more likely to be ranked higher, but the relation of preferences cannot be explicitly reflected by the corresponding representations of users and answers. Thus,
2 RELATED WORK

We discuss the related work in four-fold.

2.1 Community Question Answering (CQA)

CQA platforms enable people to seek and share knowledge effectively. For instance, Stack Overflow is a prominent CQA platform about programming, where many developers actively learn from others or share their expertise. Various CQA tasks have been extensively studied [31], like post recommendation [40] and duplicate post detection [33]. In this paper, we study five CQA tasks of the following three types: link prediction, ranking, and classification. Details of these tasks are in Section 3.1. Despite some existing works about MTL for CQA [45, 46], to the best of our knowledge, no previous MTL algorithm can solve CQA tasks from the aspect of multi-relational graphs, and the tasks considered in previous MTL works are relatively homogeneous. In contrast, our approach is the first multi-relational graph based MTL model tackling highly heterogeneous CQA tasks.

2.2 Multi-Task Learning (MTL)

MTL aims to strengthen a model’s generalizability by jointly learning multiple related tasks [5, 25, 48]. Successful applications in many domains like Natural Language Processing [21, 26, 30] have verified MTL’s effectiveness. Recent MTL models usually employ deep neural networks [21, 26]. Most of them adopt either hard parameter sharing where each task shares the same hidden layers while keeping its output layer, or soft parameter sharing where every task has its model [25]. However, most existing MTL models only undertake tasks with small heterogeneity [25, 48], like considering regression and classification of identical learning features [47], or assuming that features are different but label spaces are the same [16]. Moreover, many existing MTL algorithms are purely data-driven [20, 25, 48], which usually prevents them from capturing the label dependency across tasks due to the large heterogeneity of multi-tasks. In contrast, our MTL model efficiently performs highly heterogeneous learning tasks, where two cross-task constraints are imposed on tasks’ relationships to regularize the joint learning.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) typically perform graph representation learning by recursive neighborhood aggregation [38]. Many GNN variants [19, 42] led to significant advancement in various graph mining tasks like link prediction. Additionally, [41] conducted a systematic study on GNNs’ representational power via the connections between GNNs and Weisfeiler-Lehman (WL) graph isomorphism test [37]. Under this framework, [41] proposes Graph Isomorphism Network (GIN) which provably accomplishes the utmost discriminative power among all kinds of GNNs. Different from the above GNNs which only consider homogeneous graphs, several variants [6, 14, 27, 36, 39, 44, 49] have been designed for multi-relational graphs. Although GNNs have demonstrated superb learning abilities, few works adopt GNNs in MTL models [34, 35]. Hence, we devise an extension of GIN termed Multi-Relational GIN as one of the main components of our MTL model to facilitate multi-relational graph representation learning. Besides GIN, our framework can be easily extended to incorporate other GNN architectures to further boost performance.

2.4 Constraint Learning

Constraint learning aims to improve a model’s performance by incorporating domain knowledge as constraints. One popular constraint learning framework is the cannot-link and must-link modeling, where constraints are typically based on ground-truth labels [2]. Another powerful scheme is the Constrained Conditional Models (CCM) [7] where learning is separated from the knowledge-aware inference. Furthermore, Posterior Regularization (PR) [10] embodies knowledge by a joint learning and inference method. However, few works apply constraints on tasks’ relationships to enhance MTL. Thus, in our MTL model, we impose cross-task constraints on tasks’ relationships, which is shown to be effective.
3 PRELIMINARIES

In this section, we describe the tasks and frequently used notations. Figure 1 depicts the generic framework for building the multi-relational graph from a CQA raw dataset, and how HMT-GIN performs MTL on the built dataset.

3.1 Tasks Description

We divide all tasks into the following three types: link prediction, ranking, and classification.

3.1.1 Link Prediction. Link prediction predicts whether an edge exists between a given node pair, where some of the existing edges are hidden from the input graph. Regarding CQA, link prediction can benefit various applications like recommendations. Here are the two link prediction tasks: (1) Tag recommendation: given a question-tag pair, predict whether the tag belongs to that question; (2) Duplicate question detection: given a question-question, predict whether the tag belongs to that question.

3.1.2 Ranking. A ranking task obtains an input item list for a user to quickly identify the desired item(s). Since many CQA platform questions have many associated answers, generating a personalized ranking of the answers can greatly save users’ time. Hence, we define a task called answer recommendation as follows: given a question with at least eight answers, provide a ranking of all its answers such that the accepted answer has a high rank. Each sample contains a question index, a list its associated answers’ indices, and the list positional index of the accepted answer as the label.

3.1.3 Classification. A classification task categorizes a certain node attribute. Since some attributes like questions’ score attribute might reveal crucial node properties, categorization of them may help identify important nodes. Here are the two link prediction tasks: (1) Answer score classification: classify an answer score into one of the integers in [0, 3], where a higher value means higher score attribute; (2) User reputation classification: classify a user’s reputation into one of the integers in [0, 4], where a higher value indicates higher reputation attribute. During the data pre-processing, labels for each target attribute are generated in advance by dividing the values into different intervals, and the corresponding attribute is masked in the dataset.

3.2 Notations

Following notations are used throughout the remaining paper: the directed and labeled input multi-relational graph is denoted as \( G = (\mathcal{V}, \mathcal{E}) \) with a node type mapping function \( \phi : \mathcal{V} \to \mathcal{N} \), and edge type mapping function \( \psi : \mathcal{E} \to \mathcal{R} \), where each node \( v \in \mathcal{V} \) has one particular node type \( \phi(v) \in \mathcal{N} \), and each edge \( e \in \mathcal{E} \) has one particular edge type \( \psi(e) \in \mathcal{R} \), and \( \mathcal{R} \) does not include the self-loop edge type. Additionally, we use Multi-Layer Perceptron (MLP) or MLP for a feedforward neural network with zero or more layers. Moreover, we use \( \sigma \) for activation functions like RELU. In addition, we use \( BN \) for Batch Normalization (BN) [15]. Furthermore, we use \( f \) for sigmoid function. We also use \( \circ \) for constructing the network by stacking different layers.

4 METHODOLOGY

We first present MRGIN. Then we explain all task-specific output layers. Thereafter, we explain the cross-task constraints on tasks’ relationships. Finally, we describe the whole MTL algorithm.

4.1 Multi-Relational Graph Isomorphism Network (MRGIN)

We propose MRGIN which, together with skip connections [11], embeds the input multi-relational graph. MRGIN is inspired by GIN [41] that has been theoretically shown to have the maximum discriminative power among all types of GNNs. Specifically, let \( d^{(l)} \) be the dimension of \( (l') \)-th MRGIN layer’s node representation. The representation of a node \( v_i \in \mathcal{V} \) in \( (l+1) \)-th MRGIN layer, denoted as \( h_i^{(l+1)} \in \mathbb{R}^{d^{(l+1)}} \), is computed as:

\[
    h_i^{(l+1)} = \sigma^{(l+1)} \circ BN^{(l+1)} \circ MLP^{(l+1)} \circ \sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} W_r^{(l+1)} h_j^{(l)} + (1 + e^{(l+1)}) W_0^{(l+1)} h_i^{(l)}, \tag{1}
\]

where \( N_i^r \) is the set of neighbor indices of node \( v_i \) under edge type \( r \in \mathcal{R} \), and \( h_j^{(l)} \in \mathbb{R}^{d^{(l)}} \) as well as \( h_i^{(l)} \in \mathbb{R}^{d^{(l)}} \) are the representations of nodes \( v_j \) and \( v_i \) respectively in \( l \)-th MRGIN layer, and both \( W_r^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}} \) and \( W_0^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}} \) are type-specific transformation matrices, and \( e^{(l+1)} \) is a scalar which is either trainable or fixed. Updating one layer is to concurrently evaluate Equation (1) for all nodes.

Briefly speaking, MRGIN aggregates transformed neighboring nodes’ representations by sum pooling. Moreover, every node’s feature vector is scaled by \( 1 + e^{(l)} \) before being aggregated into its representation in the next layer. The vector produced by the summation is then processed via a non-linear activation function, followed by an MLP, a BN, and finally another non-linear activation function. One crucial distinction between MRGIN and GIN is that the transformation matrices in the aggregation of MRGIN depend on...
edges’ orientations and types, which enables MRGIN to exploit the input graph’s structural and relational information.

4.2 Task-Specific Output Layers (TSOLs)

We design TSOLs for task types in Section 3.1 as follows:

4.2.1 Link Prediction. We associate each link prediction task \( t \) with a diagonal matrix \( D_t \in \mathbb{R}^{k \times k} \) initialized by a standard normal distribution, where \( k \) is the dimension of the input node embeddings for TSOLs. Given a node pair \((u_i, v_j)\) for a task \( t \), we follow the DisMult algorithm [43] to calculate the link prediction score \( S_{ij} \) as follows:

\[
S_{ij} = h_i^T D_t h_j \tag{2}
\]

where \( h_i^t \) and \( h_j^t \in \mathbb{R}^k \) is the input node embeddings for TSOLs of \( v_i \) and \( v_j \in \mathcal{V} \) respectively. Suppose the type of the target link in task \( t \) is \( r_t \), define a set of node pairs as follows:

\[
\mathcal{H}_t = \{ (u_i, v_j) | v_i \in \mathcal{V}, v_j \in \mathcal{V}, \exists e \in E \text{ from } v_i \text{ to } v_j \text{ such that } \psi(e) = r_t \}. \tag{3}
\]

Let \( \mathcal{H}_t' \) be the set of node pairs obtained by the negative sampling procedure mentioned in Section 3.1 for task \( t \). Then task \( t \)’s loss \( L_t \) is:

\[
L_t = -\frac{1}{|\mathcal{H}_t| + |\mathcal{H}_t'|} \sum_{(u_i, v_j) \in \mathcal{H}_t \cup \mathcal{H}_t'} [w \cdot y_{ij} \cdot \log(f(S_{ij})) + (1 - y_{ij}) \cdot \log(1 - f(S_{ij}))], \tag{4}
\]

where \( w \) is the weight of positive samples (i.e., node pairs in \( \mathcal{H}_t \)), and \( y_{ij} = 1 \) if \((u_i, v_j) \in \mathcal{H}_t \) and \( y_{ij} = 0 \) if \((u_i, v_j) \in \mathcal{H}_t' \).

4.2.2 Ranking. Regarding ranking task \( t \), the parameters of its TSOL include \( w_t \in \mathbb{R}^{2k} \) and \( b_t \in \mathbb{R} \). Denote the \( i \)-th input sample as \((Q_i, A(Q_i))\) where \( Q_i \) is a question index and \( A(Q_i) \) is a list containing the indices of the question’s answers. Denote the sample’s label as \( y_i \). Motivated by RankNet [4], we compute the loss term of \( i \)-th sample, denoted as \( L_t^i \), as follows: (1) Retrieve the input node embeddings for TSOLs by \( Q_i \) and \( A(Q_i) \); (2) Concatenate the question embedding with each of the answer embeddings. Denote the concatenated embeddings as \( M_t^i \in \mathbb{R}^{2k \times |A(Q_i)|} \); (3) Compute the ranking score of answer in the \( j \)-th position of \( A(Q_i) \), denoted as \( R_j \), by

\[
R_j = \sigma(w_t^T M_t^i[:, j] + b_t), \tag{5}
\]

where \( M_t^i[:, j] \) is the \( j \)-th column of \( M_t^i \) matrix; (4) Calculate the loss term as

\[
L_t^i = -\frac{1}{|A(Q_i)|} \sum_{j=1,j \neq y_i}^{|A(Q_i)|} [\log(f(R_{y_i} - R_j))]. \tag{6}
\]

The loss of this task denoted as \( L_t \) is the average of all input samples’ loss terms. Since Equation (6) is essentially a pairwise ranking loss, minimizing \( L_t \) amounts to maximizing the difference between the ranking score of the accepted answer and those of the other answers in each input sample, which results in recommendations where the ranks of the accepted answers tend to be high.

4.2.3 Classification. Regarding a classification task \( t \), we use a MLP with one layer as its TSOL. The weight and bias of this MLP are \( W_t \in \mathbb{R}^{K_t \times k} \) and \( b_t \in \mathbb{R}^{K_t} \) respectively, where \( K_t \) is the number of possible classes in task \( t \). Given an input sample, which is a node embedding vector \( h_i^t \in \mathbb{R}^k \), the TSOL first computes the logit vector, denoted as \( x_i \in \mathbb{R}^{K_t} \), by a linear transformation:

\[
x_i = W_t h_i^t + b_t. \tag{7}
\]

Then the loss of task \( t \), denoted as \( L_t \), is calculated by softmax cross-entropy:

\[
L_t = -\frac{1}{N_t} \sum_{i=1}^{N_t} \left( \log \left( \frac{\exp(x_i[y_i])}{\sum_{j=1}^{K_t} \exp(x_i[j])} \right) \right), \tag{8}
\]

where \( N_t \) is the number of input samples and \( y_i \) is the class label of the \( i \)-th sample.

4.3 Cross-task Constraints

Besides data-driven learning, our model can be equipped with domain knowledge about tasks’ relationships via incorporating several cross-task constraints into the objective function to regularize the joint learning. In this paper, we design two cross-task constraints as a demonstration of our constraint learning framework. Apart from these constraints, our framework can easily incorporate more cross-task constraints exploiting the domain knowledge about tasks’ relationships by simply adding extra constraint loss functions into the objective.

Each of the two cross-task constraints relies on one of the following two assumptions based on the domain knowledge about CQA. Given two answers \( A_1 \) and \( A_2 \) of a question:

1. **Assumption 1**: if \( A_1 \)’s score attribute is higher than that of \( A_2 \), then \( A_1 \) is more likely to be accepted;
2. **Assumption 2**: if \( A_1 \)’s owner’s reputation attribute is higher than that of \( A_2 \), then \( A_1 \) is more likely to be accepted.

Assumption 1 is reasonable as an answer with higher score attribute usually has higher quality. Similarly, Assumption 2 is also legitimate since a user with higher reputation attribute typically has higher capability, which implies that the user’s answer tends to be better. Hence, we propose the following two cross-task constraints.

The constraint of assumption 1 encourages answerer(s) predicted to have the highest score attribute in each sample to rank highly in the recommendation list. Figure (2a)-(2b) illustrate an example of computing this constraint loss. The general idea is: (1) For each input sample (i.e., a question with all of its answers), use the output layer of answer score classification to compute all answers’ logits; (2) Use the output layer of answer recommendation to get the ranking score(s) of answer(s) with the highest logit; (3) Compute negative log-likelihood functions based on these ranking scores from all input samples, which are then averaged to get the overall constraint loss function \( C_t \).

Specifically, we store a sample list \( I_t \) before training, where each sample is a question index with a list of the indices of the question’s answers. Let the \( i \)-th sample be \((Q_i, A(Q_i))\) where \( Q_i \) is the question index and \( A(Q_i) \) is the answer indices list. This sample’s loss term is calculated as follows: (1) Retrieve the corresponding node embeddings by \( Q_i \) and \( A(Q_i) \); (2) Calculate the logit values of the...
(a) Constraint 1 step (1): compute logits via answer classification output layer and select the answer with highest logit.

(b) Constraint 1 step (2), (3): compute the ranking score(s) of answer(s) with the highest logit, then compute negative log-likelihoods which are averaged to get the overall constraint loss.

(c) Constraint 2 step (1), (2): find owner users and compute their logits via user classification output layer.

(d) Constraint 2 step (3), (4): compute the ranking score(s) of answers whose owner user(s) has the highest logit, then compute negative log-likelihoods which are averaged to get the overall constraint loss.

Figure 2: Examples of computing the proposed two cross-task constraints’ loss functions. Figures (a)-(b) illustrate the 1st constraint, while figures (c)-(d) illustrate the 2nd constraint.

The first constraint loss $C_1$ is the average of loss terms of all samples in $I_1$. Since minimizing Equation (9) amounts to maximizing the ranking scores of answers corresponding to $I_1$, reducing $C_1$ will encourage answer(s) which is predicted to have the highest value of score attribute in each sample to rank highly, which enhances the consistency between the answer score classification and answer recommendation tasks.

The constraint of assumption 2 encourages the answer(s) whose user is predicted to have the highest reputation in each sample to have higher rank in the recommendation list. Figure (2c)-(2d) illustrate an example of computing this constraint loss. The general idea is: (1) For each input sample (i.e., a question with all of its answers), find the answers’ owner users; (2) Use the output layer of user classification to compute the logits of all these users; (3) Use the output layer of answer recommendation to get the ranking score(s) of answer(s) whose owner(s) has the highest logit; (4) Compute negative log-likelihood functions based on these ranking scores from all samples as in Equation 10 (the notations are similar to those in Equation 9 for the first constraint), which are then averaged to get the overall constraint loss $C_2$.

Because minimizing Equation (10) is equivalent to maximizing the ranking scores of answers corresponding to $I_2$, decreasing $C_2$ will provide the answer(s) whose user is predicted to have the highest reputation in each sample with more chance of having a high rank in the recommendation list, which strengthens the consistency between the user reputation classification and answer recommendation tasks.

Admittedly, this approach of designing constraints is somewhat ad-hoc. Nonetheless, these cross-task constraints, which reflect human’s domain knowledge of the tasks, not only are lucid and easy to implement but also significantly boost our model’s performance as verified in the experiments.
We obtained the Stack Overflow raw dataset from the Stack Exchange Data Dump. We then performed data cleaning steps like removing attributes with too many missing values. Afterward, we constructed a million-scale multi-relational graph consisting of questions with at least eight answers, all other types of nodes that have either directly or indirectly connected to these questions, and associated edges. The graph has 4 node types and 22 edge types including self loop edge types for all node types and all reverse edge types.

4.4 Heterogeneous Multi-Task Graph Isomorphism Network (HMTGIN)

Finally, we illustrate our MTL algorithm called HMTGIN. Suppose the input multi-relational CQA graph consists of graph structure and initial node embeddings, and there are \( T \) tasks and \( M \) cross-task constraints. The training process of HMTGIN is as follows: (1) The MRGIN model defined in Section 4.1 learns new node embeddings by taking in the initial node embeddings and then performing neighborhood aggregations based on the graph structure. The new node embeddings are termed residual node embeddings; (2) Skip-connection technique is adopted by performing node-wise concatenation between the initial node embeddings and the residual node embeddings. We call these concatenated embeddings as generated node embeddings; (3) The generated node embeddings are shared by every TSOL to calculate corresponding loss \( L_t(t \in \{1, 2, ..., T\}) \) as described in Section 4.2; (4) Cross-task constraint losses \( C_1, C_2, ..., C_M \) are computed using respective generated node embeddings as described in Section 4.3; (5) Compute the total loss \( L' \) as:

\[
L' = \frac{1}{T} \sum_{t=1}^{T} \alpha_t L_t + \sum_{i=1}^{M} \beta_i C_i
\]

where \( \alpha_t (t \in \{1, 2, ..., T\}) \) and \( \beta_i (i \in \{1, 2, ..., M\}) \) are pre-defined constant real numbers. (6) Update HMTGIN’s learnable parameters by back-propagation with respect to the total loss \( L' \); (7) Repeat Step (1) - (6) until the termination condition is met. In evaluation, the generated node embeddings after the last training epoch are shared across different TSOLs to make corresponding predictions. A generic overview of our model is shown in Figure 3.

5 EXPERIMENTS

In this section, we present our experiments.

5.1 Dataset

We obtained the Stack Overflow raw dataset from the Stack Exchange Data Dump. We then performed data cleaning steps like removing attributes with too many missing values. Afterward, we constructed a million-scale multi-relational graph consisting of questions with at least eight answers, all other types of nodes that are either directly or indirectly connected to these questions, and associated edges. The graph has 4 node types and 22 edge types including self loop edge types for all node types and all reverse edge types.

| Type     | Cardinality |
|----------|-------------|
| Questions| 108,113     |
| Answers  | 1,212,308   |
| Users    | 773,517     |
| Tags     | 55,663      |

Figure 4: Schema of the Stack Overflow dataset.

Figure 3: Generic overview of HMTGIN. TSOL (i) denotes the task-specific output layer of \( i \)-th task.

Table 1: Cardinalities of all node types.

\(^1\)Any two edges which have either different source node types or different destination node types are considered to be of different edge types (e.g., ‘owner_of’ edge type between Users and Questions are considered to be different from that between Users and Answers).

\(^2\)Reverse edge types are omitted for conciseness.
We split each task’s dataset into training, development, and test sets. We compare our model with the following algorithms: (1) WDL [8]; (2) NCF [12]; (3) CDSSM [29]; (4) MalSTM [22]; (5) Text-CNN [17]; (6) BiLSTM [28]; (7) TransE [3]; (8) COMPGCN [32]; (9) KBGAT [23]; (10) BERT (Large) [9]. We do not include any of the existing MTL models for CQA in the experiment since none of them can jointly learn the highly heterogeneous tasks targeted in this paper.

Baseline models for the ranking task are extended to incorporate the loss function of RankNet [4] to boost performance. Besides the above baselines, to verify the effectiveness of MRGIN, we also compare HMTGIN with one of its variants termed Heterogeneous Multi-Task Graph Convolutional Networks (HMTGCN) which is the same as HMTGIN except that it replaces the MRGIN component of HMTGIN with RGCNs. The baseline algorithms and evaluation metrics for all tasks are included in Table 2, where F1 means F1 score, HR@3 means Hit Ratio in top-3 list, NDCG@3 means Normalized Discounted Cumulative Gain in top-3 list, and Macro F1 means macro F1 score for multi-class classification.5 The word embeddings are fixed during training for CDSSM, MalSTM, Text-CNN, BiLSTM, and BERT.

We abbreviate task names as follows: ‘TR’ means tag recommendation; ‘DQQD’ denotes duplicate question detection; ‘AR’ means answer recommendation; ‘ASC’ denotes answer score classification; ‘URC’ means user reputation classification. Regarding each task, we train and test the corresponding model instances with their respective best hyper-parameters under three different random seeds. The mean test scores and the standard deviations of all models are shown in Table 3, 4, 5, 6, and 7 respectively. The best mean score in each task for every corresponding evaluation metric is marked in bold.

The hyper-parameters we have tuned for HMTGIN as follows, where the values used in the best configuration of HMTGIN are shown in bold.

(1) number of hidden layer: 0, 1, 2, 3;
(2) hidden dimension(s) : 8, 16, 32, 64;
(3) whether the \( \epsilon \) coefficient in Equation 1 is trainable or not: yes, no;
(4) dropout rate in all MRGIN layer: 0, 0.1, 0.2, 0.3;
(5) number of mlp layer in each MRGIN layer: 0, 1, 2, 3, 4.

We train each HMTGIN instance for 135 epochs, where the checkpoint is saved whenever the average score increases. The coefficients of both cross-task constraints are 1. The coefficients associated with the losses of both duplication question detection and answer score classification are 7, whereas the others are 1. The model is trained using Adam [18] algorithm in full batch. The \( \sigma \) activation function in Equation 1 and Equation 5 is Leaky ReLU. The value of the \( \epsilon \) coefficient in Equation 1 is 0. The learning rate is reduced by 0.5 in every 50 epochs.

5.4 Performance Comparison

We abbreviate task names as follows: ‘TR’ means tag recommendation; ‘DQQD’ denotes duplicate question detection; ‘AR’ means answer recommendation; ‘ASC’ denotes answer score classification; ‘URC’ means user reputation classification. Regarding each task, we train and test the corresponding model instances with their respective best hyper-parameters under three different random seeds. The mean test scores and the standard deviations of all models are shown in Table 3, 4, 5, 6, and 7 respectively. The best mean score in each task for every corresponding evaluation metric is marked in bold.

In summary, HMTGIN outperforms all baselines in all tasks, where the improvements are up to 16.9%, which distinctly demonstrates its effectiveness.

Regarding link prediction, the improvements in accuracy are up to 12%, and the improvements in F1 score are up to 13.3%. Regarding ranking, the improvements are up to 11.3% and 12.9% in HR@3 and NDCG@3 respectively. Such considerable improvements indicate that the proposed cross-task constraints are significant. Regarding classification, all HMTGIN’s scores in URC task substantially exceed those of the corresponding best baselines (the improvements over

---

Table 2: Baselines and evaluation metrics for all tasks.

| Tasks           | Baselines         | Metrics       |
|-----------------|-------------------|---------------|
| Tag             | WDL, NCF, TransE  | Accuracy, F1 |
| Recommendation  | COMPGCN, KBGAT    |               |
| Duplicate Question Detection | CDSSM, MalSTM, TransE | Accuracy, F1 |
| Detection       | COMPGCN, KBGAT    |               |
| Answer          | WDL, NCF          | HR@3, NDCG@3 |
| Recommendation  | COMPGCN           |               |
| Answer Score    | Text-CNN, BiLSTM  | Accuracy, F1  |
| Classification  | COMPGCN, BERT     | Macro F1      |
| User Reputation | Text-CNN, BiLSTM  | Accuracy, F1  |
| Classification  | COMPGCN, BERT     | Macro F1      |

We omit micro F1 score as it always has the same value as accuracy does in this setting.

---

Table 3: Performance comparison in the TR task.

| Models | Metrics        | Accuracy | F1    |
|--------|----------------|----------|-------|
| WDL    | 0.870 ± 0.002  | 0.786 ± 0.001 |
| NCF    | 0.871 ± 0.001  | 0.786 ± 0.001 |
| TransE | 0.879 ± 0.000  | 0.795 ± 0.000 |
| KBGAT  | 0.895 ± 0.002  | 0.810 ± 0.001 |
| HMTGIN | 0.911 ± 0.002  | 0.866 ± 0.002 |
|        | **0.921 ± 0.000** | **0.878 ± 0.000** |

The numbers to the left of ‘*’ symbols are the mean scores and the numbers to the right of ‘*’ symbols are the corresponding standard deviations.
### Table 4: Performance comparison in the DQD task.

| Models  | Accuracy   | F1       |
|---------|------------|----------|
| CDSSM   | 0.795 ± 0.004 | 0.656 ± 0.003 |
| MaLSTM  | 0.794 ± 0.003 | 0.601 ± 0.002 |
| TransE  | 0.683 ± 0.001 | 0.552 ± 0.001 |
| COMPGCN | 0.702 ± 0.003 | 0.583 ± 0.002 |
| KBGAT   | 0.721 ± 0.002 | 0.591 ± 0.002 |
| HMTGCN  | 0.773 ± 0.001 | 0.645 ± 0.002 |
| HMTGIN  | **0.803 ± 0.001** | **0.685 ± 0.001** |

### Table 5: Performance comparison in the AR task.

| Models | HR@3     | NDCG@3  |
|--------|----------|---------|
| WDL    | 0.624 ± 0.001 | 0.49 ± 0.001 |
| NCF    | 0.631 ± 0.001 | 0.498 ± 0.001 |
| COMPGCN | 0.697 ± 0.001 | 0.583 ± 0.002 |
| HMTGCN | 0.717 ± 0.002 | 0.589 ± 0.003 |
| HMTGIN | **0.737 ± 0.000** | **0.619 ± 0.000** |

### Table 6: Performance comparison in the ASC task.

| Models | Accuracy   | Macro F1 |
|--------|------------|----------|
| Text-CNN | 0.421 ± 0.001 | 0.423 ± 0.002 |
| BiLSTM  | 0.435 ± 0.004 | 0.438 ± 0.005 |
| COMPGCN | 0.331 ± 0.002 | 0.342 ± 0.001 |
| BERT    | 0.442 ± 0.001 | 0.438 ± 0.001 |
| HMTGCN  | 0.434 ± 0.002 | 0.429 ± 0.002 |
| HMTGIN  | **0.444 ± 0.001** | **0.439 ± 0.002** |

### Table 7: Performance comparison in the URC task.

| Models | Accuracy   | Macro F1 |
|--------|------------|----------|
| Text-CNN | 0.395 ± 0.003 | 0.393 ± 0.003 |
| BiLSTM  | 0.405 ± 0.001 | 0.396 ± 0.003 |
| COMPGCN | 0.314 ± 0.001 | 0.302 ± 0.001 |
| BERT    | 0.459 ± 0.001 | 0.436 ± 0.001 |
| HMTGCN  | 0.423 ± 0.002 | 0.417 ± 0.001 |
| HMTGIN  | **0.483 ± 0.002** | **0.470 ± 0.001** |

accuracy and macro F1 are 2.4% and 3.4% respectively), whereas in ASC task, HMTGIN only slightly outperforms the corresponding best baselines. This might result from that the best baseline is already powerful enough for ASC task, which leads to a small room for further improvement. By comparing the performance of HMTGIN and HMTGCN, we can see that HMTGIN consistently outperforms HMTGCN in all tasks, where the improvements are up to 6%, and in two out of the five tasks, the improvements are at least 3%. This result shows that GIN is more powerful than GCN.

### 5.5 Ablation Studies

We conduct the following ablation studies: (1) comparing Single Task Learning (STL) settings with MTL settings; (2) examining the influence of the cross-task constraints. (3) parameter sensitivity study on the number of MLP layers in each MRGIN layer. Every numerical result is the average over three random seeds.

#### 5.5.1 STL Settings VS MTL Settings

The STL scores together with the MTL ones are shown in Figure 5, and the x-axis contains the task names and respective metrics.

Generally speaking, the MTL instance outperforms all of its STL counterparts for all evaluation metrics (i.e., there is no negative transfer), where the improvements are up to 8%, which justifies the effectiveness of our MTL strategy. In particular, the MTL instance surpasses the STL instances by a large margin in TR (3.8% and 4.7% in accuracy and F1 score respectively), DQD (2.8% and 2.2% in accuracy and F1 score respectively), and AR (7.1% and 8% in HR@3 and NDCG@3 respectively). By contrast, the performance gain in the ASC task (0.7% in both accuracy and macro F1 score) and URC task (1.2% and 1.1% in accuracy and macro F1 score respectively) is not very noticeable. Such a distinction indicates that learning signals from the MTL mechanism are more crucial for the first three tasks than the last two tasks.

#### 5.5.2 Role of Cross-task Constraints

We conduct experiments on the following settings of cross-task constraints to examine their significance, where the results are included in Figure 6: (1) No constraint means removing both constraints described in Section 4.3; (2) Constraint 1 only denotes retaining only constraint 1, and similarly for Constraint 2 only; (3) Both 0.5 denotes keeping both constraints, and setting both their coefficients as 0.5, and similarly for Both 1 and Both 2.

We only show and discuss the F1 scores for link prediction tasks, NDCG@3 score for ranking task, and macro F1 scores for classification task here due to space limitation, and the pattern of the scores which are not included here is almost the same. The x-axis is similar to that of Figure 5, where the ‘Average’ includes the average value over all the five shown scores of the instance under each of the above five settings.

In general, Both 1 instance achieves the best average score, which shows that the proposed cross-task constraints are beneficial to the overall performance. Moreover, keeping both constraints almost always improves performance in the tasks related to the constraints (i.e., AR, ASC, and URC), except in a few cases where there are still 1Configuration of the HMTGIN instance in Section 5.4 corresponds to this setting.
Fig. 6: Comparison between different settings of HMT-GIN’s cross-task constraints.

Fig. 7: Comparison between different numbers of the MLP layer in each MRGIN layer.

some settings that retain both constraints and have the best scores for the corresponding evaluation metrics. For instance, Both 0.5, Both 1, and Both 2 instances all outperform the remaining settings in both evaluation metrics for AR, where the improvements for NDCG@3 range from 2.3% to 25% respectively. These enhancements are attributed to the cross-task constraints which strengthen the consistency between related tasks, as illustrated in Section 4.3. Specifically, No constraint and Constraint 1 only instances consistently perform worse than the setting Constraint 2 only in URC (the average difference for macro F1 is 3.2%), which empirically justifies the analyses in Section 4.3 as the constraint 2 is established on the relationship between URC and AR. In addition, Constraint 1 only instance surpasses both Constraint 2 only and No constraint instances in ASC (the average improvements in macro F1 is 8.9%), which is also consistent with the explanation in Section 4.3 since the constraint 1 reinforces the coherence between AR and ASC. Another interesting observation is that the average score of Both 0.5 instance is quite close to those of Constraint 1 only, Constraint 2 only and No Constraint instances, where the improvements range from 0.1% to 2.1%. This might be because the small weights of both coefficients weaken the regularization power of the constraints. However, the average score of the Constraint 2 only instance is worse than that of the Constraint 1 only instance, although the gap is tiny (0.3%). This is probably because the high weights of both coefficients somewhat impede the optimization of losses. Accordingly, choosing appropriate values of both constraints’ coefficients is critical.

5.5.3 Parameter sensitivity study on the number of MLP layers in each MRGIN layer. The performance with different number of MLP layer in each MRGIN layer is shown in Figure 7, where the 0 is the instance with no MLP module, the 1 is the instance with 1 MLP layer, and so on.

To summarize, the presence of MLP layer greatly boosts the performance (the average improvements are up to 6.9%), which can be attributed to the universal approximation theorem [13] ensuring that MLP can learn the composition of the multiset functions that allows GIN to have the maximum discriminative power among all GNN variants [41]. Furthermore, the instance with one MLP layer has the best average score (i.e., 64.8%), even though the other instances with MLP module are only slightly weaker than it (the largest margin in the average score is only 0.3%). This indicates that altering the number of MLP layer in each MRGIN layer of HMTGIN have limited effect when there is at least one MLP layer in each MRGIN layer. Moreover, increasing the number of MLP layer does not exhibit any generally monotonic trend in the average scores (the changes from zero MLP layer to one MLP layer, from one to two MLP layers, from two MLP layers to three MLP layers, and from three MLP layers to four MLP layers are 6.9%, –0.3%, 0.1% and –0.1% respectively). The performance gain offered by the MLP module is most distinct in TR, where the improvements from the instance without MLP layer to instances with MLP layer are up to 32.7% and 29.0% for accuracy and F1 respectively. Such considerable improvements might be since the MLP layer is highly effective in capturing the tags’ semantics. By contrast, the instances with MLP module tend to perform worse compared with those without MLP module in classification tasks, especially in ASC, where the instance without MLP layer outperforms all the instances with MLP layer in both evaluation metrics (the score differences range from 0.3% to 1.0% and 0.7% to 1.4% for accuracy and macro F1 respectively). Such results suggest that MLP is not very helpful for distinguishing the data characteristics in MTL.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose a multi-relational graph based MTL model named HMTGIN which efficiently tackles CQA with considerable task and graph heterogeneity. To the best of our knowledge, HMTGIN is the first MTL model solving CQA tasks from the angle of multi-relational graphs. To evaluate the effectiveness of HMTGIN, we build a novel million-scale multi-relational graph CQA dataset. Experiments of five tasks demonstrate that HMTGIN surpasses all baselines in all tasks. Further ablation studies manifest the substantial role of the proposed cross-task constraints and MTL strategy. An interesting future direction would be investigating how to develop a more principled way to incorporate the domain knowledge by imposing cross-task constraints on multi-relational graph based MTL algorithms for CQA tasks.

7 ACKNOWLEDGEMENT

The authors of this paper were supported by the NSFC Fund (U20B2053) from the NSFC of China, the RIF (R6020-19 and R6021-20) and the GRF (16211520) from RGC of Hong Kong, the MHKJFS (MHP/001/19) from ITC of Hong Kong, with special thanks to the WeChat-HKUST WHAT Lab on Artificial Intelligence Technology.

\(^8\)The 1 instance corresponds to the configuration in Section 5.4.
