Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks

Qingsong Lv††, Ming Ding††, Qiang Liu‡, Yuxiang Chen†, Wenzheng Feng†, Siming He‡‡, Chang Zhou‡, Jianguo Jiang‡, Yuxiao Dong¶, Jie Tang‡‡§
† Tsinghua University • Chinese Academy of Sciences ‡ Alibaba Group ‡ University of Pennsylvania ¶ Microsoft
{lqs19,dm18,chenyuxi18,fwz17}@mails.tsinghua.edu.cn, {liuqiang,jiangjianguo}@iie.ac.cn, ericzhou.zc@alibaba-inc.com, siminghe@seas.upenn.edu, ericdongyx@gmail.com, jietang@tsinghua.edu.cn

ABSTRACT
Heterogeneous graph neural networks (HGNNs) have been blossoming in recent years, but the unique data processing and evaluation setups used by each work obstruct a full understanding of their advancements. In this work, we present a systematical reproduction of 12 recent HGNNs by using their official codes, datasets, settings, and hyperparameters, revealing surprising findings about the progress of HGNNs. We find that the simple homogeneous GNNs, e.g., GCN and GAT, are largely underestimated due to improper settings. GAT with proper inputs can generally match or outperform all existing HGNNs across various scenarios. To facilitate robust and reproducible HGNN research, we construct the Heterogeneous Graph Benchmark (HGB)†, consisting of 11 diverse datasets with three tasks. HGB standardizes the process of heterogeneous graph data splits, feature processing, and performance evaluation. Finally, we introduce a simple but very strong baseline —Simple-HGN—which significantly outperforms all previous models on HGB—to accelerate the advancement of HGNNs in the future.

CCS CONCEPTS
• Computing methodologies → Neural networks; • Mathematics of computing → Graph algorithms.

KEYWORDS
Graph Neural Networks; Heterogeneous Graphs; Graph Representation Learning; Graph Benchmark; Heterogeneous Networks

ACM Reference Format:

1 INTRODUCTION
As graph neural networks (GNNs) [2, 21] have already occupied the centre stage of graph mining research within recent years, the researchers begin to pay attention to their potential on heterogeneous graphs (a.k.a., Heterogeneous Information Networks) [8, 12, 19, 36, 40, 43]. Heterogeneous graphs consist of multiple types of nodes and edges with different side information, connecting the novel and effective graph-learning algorithms to the noisy and complex industrial scenarios, e.g., recommendation.

To tackle the challenge of heterogeneity, various heterogeneous GNNs (HGNNs) [36, 40, 43] have been proposed to address the relevant tasks, including node classification, link prediction, and knowledge-aware recommendation. Take node classification for example, numerous HGNNs, such as HAN [36], GTN [43], RSHN [45], HetGNN [44], MAGNN [12], HGT [20], and HetSANN [17] were developed within the last two years.

Despite various new models developed, our understanding of how they actually make progress has been thus far limited by the unique data processing and settings adopted by each of them. To fully picture the advancements in this field, we comprehensively reproduce the experiments of 12 most popular HGNN models by using the codes, datasets, experimental settings, hyperparameters released by their original papers. Surprisingly, we find that the results generated by these state-of-the-art HGNNs are not as exciting as promised (Cf. Table 1), that is:

1. The performance of simple homogeneous GNNs, i.e., GCN [21] and GAT [32], is largely underestimated. Even vanilla GAT can outperform existing HGNNs in most cases with proper inputs.
2. Performances of some previous works are mistakenly reported due to inappropriate settings or data leakage.

Our further investigation also suggests:

(3) Meta-paths are not necessary in most heterogeneous datasets.
(4) There is still considerable room for improvements in HGNNs.

In our opinion, the above situation occurs largely because the individual data and experimental setup by each work obstructs a fair and consistent validation of different techniques, thus greatly hindering the advancements of HGNNs.

To facilitate robust and open HGNN developments, we build the HETEROGENEOUS GRAPH BENCHMARK (HGB). HGB currently contains 11 heterogeneous graph datasets that vary in heterogeneity (the number of node and edge types), tasks (node classification, link prediction, and knowledge-aware recommendation), and domain
(e.g., academic graphs, user-item graphs, and knowledge graphs). HGB provides a unified interface for data loading, feature processing, and evaluation, offering a convenient and consistent way to compare HGNN models. Similar to OGB [18], HGB also hosts a leaderboard (https://www.biendata.xyz/hgb) for publicizing reproducible state-of-the-art HGNNs.

Finally, inspired by GAT’s significance in Table 1, we take GAT as backbone to design an extremely simple HGNN model—Simple-HGN. Simple-HGN can be viewed as GAT enhanced by three existing techniques: (1) learnable type embedding to leverage type information, (2) residual connections to enhance modeling power, and (3) $L_2$ normalization on the output embeddings. In ablation studies, these techniques steadily improve the performance. Experimental results on HGB suggest that Simple-HGN can consistently outperform previous HGNNs on three tasks across 11 datasets, making it to date the first HGNN model that is significantly better than the vanilla GAT.

To sum up, this work makes the following contributions:

- We revisit HGNNs and identify issues blocking progress;
- We benchmark HGNNs by HGB for robust developments;
- We refine HGNNs by designing the Simple-HGN model.

## 2 PRELIMINARIES

### 2.1 Heterogeneous Graph

A heterogeneous graph [29] can be defined as $G = \{V, E, \phi, \psi\}$, where $V$ is the set of nodes and $E$ is the set of edges. Each node $\nu$ has a type $\phi(\nu)$, and each edge $e$ has a type $\psi(e)$. The sets of possible node types and edge types are denoted by $T_\nu = \{\phi(\nu) : \nu \in V\}$ and $T_e = \{\psi(e) : \forall e \in E\}$, respectively. When $|T_\nu| = |T_e| = 1$, the graph degenerates into an ordinary homogeneous graph.

### 2.2 Graph Neural Networks

GNNs aim to learn a representation vector $h_i^{(l)} \in \mathbb{R}^{dL}$ for each node $\nu$ after $L$-layer transformations, based on the graph structure and the initial node feature $h_\nu^{(0)} \in \mathbb{R}^{d_\nu}$. The final representation can serve various downstream tasks, e.g., node classification, graph classification (after pooling), and link prediction.

**Graph Convolution Network (GCN)** [21] is the pioneer of GNN models, where the $l^{th}$ layer is defined as

$$H^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)}),$$  \hspace{1cm} (1)

where $H^{(l)}$ is the representation of all nodes after the $l^{th}$ layer, $W^{(l)}$ is a trainable weight matrix, $\sigma$ is the activation function, and $\hat{A}$ is the normalized adjacency matrix with self-connections.

**Graph Attention Network (GAT)** [32] later replaces the average aggregation from neighbors, i.e., $\hat{A}H^{(l-1)}$ as a weighted one, where the weight $a_{ij}$ for each edge $(i, j)$ is from an attention mechanism as (layer mark $(l)$ is omitted for simplicity)

$$a_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^T[Wh_i||Wh_j]\right)\right)}{\sum_{k \in N_i} \exp\left(\text{LeakyReLU}\left(a^T[Wh_i||Wh_k]\right)\right)},$$  \hspace{1cm} (2)

where $a$ and $W$ are learnable weights and $N_i$ represents the neighbors of node $i$. Multi-head attention technique [31] is also used to improve the performance.

Many following works [1, 7, 11, 37] improve GCN and GAT furthermore, with focuses on homogeneous graphs. Actually, these homogeneous GNNs can also handle heterogeneous graphs by simply ignoring the node and edge types.

## 3 ISSUES WITH EXISTING HETEROGENEOUS GNNS

We analyze popular heterogeneous GNNs (HGNNs) organized by the tasks that they aim to address. For each HGNN, the analysis will be emphasized on its defects found in the process of reproducing its result by using its official code, the same datasets, settings, and hyperparameters as its original paper, which is summarized in Table 1.

### 3.1 Node Classification

#### 3.1.1 HAN [36].

Heterogeneous graph attention network (HAN) is among the early attempts to tackle with heterogeneous graphs. Firstly, HAN needs multiple meta-paths selected by human experts. Then HAN uses a hierarchical attention mechanism to capture both node-level and semantic-level importance. For each meta-path, the node-level attention is achieved by a GAT on its corresponding meta-path neighbor graph. And the semantic-level attention, which gives the final representation, refers to a weighted average of the node-level results from all meta-path neighbor graphs.

A defect of HAN is its unfair comparison between HAN and GAT. Since HAN can be seen as a weighted ensemble of GATs on many meta-path neighbor graphs, a comparison with the vanilla GAT is essential to prove its effectiveness. However, the GCN and GAT baselines in this paper take only one meta-path neighbor graph as input, losing a large part of information in the original graph, even though they report the result of the best meta-path neighbor graph.

To make a fair comparison, we feed the original graph into GAT by ignoring the types and only keeping the features of the target-type nodes. We find that this simple homogeneous approach consistently outperforms HAN, suggesting that the homogeneous GNNs are largely underestimated (See Table 1 for details).

Most of the following works also follow HAN’s setting to compare with homogeneous GNNs, suffering from the “information missing in homogeneous baselines” problem, which leads to a positive cognitive deviation on the performance progress of HGNNs.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>ACM</td>
<td>DBLP</td>
<td>ACM</td>
<td>IMDB</td>
<td></td>
</tr>
<tr>
<td>Model*</td>
<td>91.89</td>
<td>91.85</td>
<td>94.18</td>
<td>92.68</td>
<td>90.55</td>
</tr>
<tr>
<td>GCN*</td>
<td>89.31</td>
<td>89.45</td>
<td>87.30</td>
<td>91.60</td>
<td>90.55</td>
</tr>
<tr>
<td>GAT*</td>
<td>90.55</td>
<td>90.55</td>
<td>93.71</td>
<td>92.33</td>
<td>92.08</td>
</tr>
</tbody>
</table>

### 3.1.2 GTN [43].

Graph transformer network (GTN) is able to discover valuable meta-paths automatically, instead of depending on manual selection like HAN. The intuition is that a meta-path neighbor graph can be obtained by multiplying the adjacency matrices of several sub-graphs. Therefore, GTN uses a soft sub-graph selection and matrix multiplication step to generate meta-path neighbor graphs, and then encodes the graphs by GCNs.

The main drawback of GTN is that it consumes gigantic amount of time and memory. For example, it needs 120 GB memory and 12 hours to train a GTN on DBLP with only 18,000 nodes. In contrast, GCN and GAT only take 1 GB memory and 10 seconds of time.

Moreover, when we test the GTN and GAT five times using the official codes of GTN, we find from Table 1 that their average scores are not significantly different, though GTN consumes > 400x time and 120x memory of GAT.

### 3.1.3 RSHN [45].

Relation structure-aware heterogeneous graph neural network (RSHN) builds coarsened line graph to obtain edge features first, then uses a novel Message Passing Neural Network (MPNN) [13] to propagate node and edge features.

The experiments in RSHN have serious problems according to the official code. First, it does not use validation set, and just tune hyperparameters on test set. Second, it reports the accuracy at the epoch with best accuracy on test set in the paper. As shown in Table 1, our well-tuned GAT can even reach 100% accuracy under this improper setting on the AIFB and BGS datasets, which is far better than the 91.67% and 66.32% reported in the original paper.

### 3.1.4 HetGNN [44].

Heterogeneous graph neural network (HetGNN) first uses random walks with restart to generate neighbors for nodes, and then leverages Bi-LSTM to aggregate node features for each type and among types.

HetGNN has the same "information missing in homogeneous baselines" problem as HAN: when comparing it with GAT, a sampled graph instead of the original full graph is fed to GAT. As demonstrated in Table 1, GAT with correct inputs gets clearly better performance.

### 3.1.5 MAGNN [12].

Meta-path aggregated graph neural network (MAGNN) is an enhanced HAN. The motivation is that when HAN deals with meta-path neighbor graphs, it only considers two endpoints of the meta-paths but ignores the intermediate nodes. MAGNN proposes several meta-path encoders to encode all the information along the path, instead of only the endpoints.

However, there are two problems in the experiments of MAGNN. First, MAGNN inherits the "information missing in homogeneous baselines" problem from HAN, and also underperforms GAT with correct inputs.

More seriously, MAGNN has a data leakage problem in link prediction, because it uses batch normalization, and loads positive and negative links sequentially during both training and testing periods. In this way, samples in a minibatch are either all positive or all negative, and the mean and variance in batch normalization will provide extra information. If we shuffle the test set to make each minibatch contains both positive and negative samples randomly, the AUC of MAGNN drops dramatically from 98.91 to 71.49 on the Last.fm dataset.

### 3.1.6 HGT [20].

Heterogeneous graph transformer (HGT) proposes a transformer-based model for handling large academic heterogeneous graphs with heterogeneous subgraph sampling. As HGT mainly focuses on handling web-scale graphs via graph sampling strategy [14, 42], the datasets used in its paper (> 10,000,000 nodes) are unaffordable for most HGNNs, unless adapting them by subgraph sampling. To eliminate the impact of subgraph sampling techniques on the performance, we apply HGT with its official code on the relatively small datasets that are not used in its paper, producing mixed results when compared to GAT (See Table 3).

### 3.1.7 HetSANN [17].

Attention-based graph neural network for heterogeneous structural learning (HetSANN) uses a type-specific attention mechanism to aggregate graph information of each local node within the same category. It also handles graphs with heterogeneous subgraph sampling. As demonstrated in Table 1, HetSANN is reported to have promising performance in the paper.

However, the datasets and preprocessing details are not released with the official codes, and responses from its authors are not received for the submission of this work. Therefore, we directly apply HetSANN with standard hyperparameter tuning, giving un-promising results on other datasets (See Table 3).
3.2 Link Prediction

3.2.1 RGCN [28]. Relational graph convolutional network (RGCN) extends GCN to relational (multiple edge types) graphs. The convolution in RGCN can be interpreted as a weighted sum of ordinary graph convolution with different edge types. For each node \( i \), the \( l \)-th layer of convolution are defined as follows,

\[
h^{(l)}_i = \sigma \left( \sum_{r \in \mathcal{E}_r} \sum_{j \in \mathcal{N}_i} \frac{1}{c_{i,r}} W_r^{(l)}h^{(l)}_j + W_0^{(l)}h^{(l-1)}_i \right),
\]

where \( c_{i,r} \) is a normalization constant and \( W_0, W_r \) are learnable parameters.

3.2.2 GATNE [5]. General attributed multiplex heterogeneous network embedding (GATNE) leverages the graph convolution operation to aggregate the embedded embeddings from neighbors. It relies on Skip-gram to learn a general embedding, a specific embedding and an attribute embedding respectively, and finally fuses all of them. In fact, GATNE is more a network embedding algorithm than a GNN-style model.

3.3 Knowledge-Aware Recommendation

Recommendation is a main application for Heterogeneous GNNs, but most related works [9, 10, 24, 25] only focus on their specific industrial data, resulting in non-open datasets and limited transferability of the models. Knowledge-aware recommendation is an emerging sub-field, aiming to improve recommendation by linking items with entities in an open knowledge graph. In this paper, we mainly survey and benchmark models on this topic.

3.3.1 KGCN [34] and KGNNS-L [33]. KGCN enhances the item representation by performing aggregations among its corresponding entity neighborhood in a knowledge graph. KGNNS-L further poses a label smoothness assumption, which posits that similar items in the knowledge graph are likely to have similar user preference. It adds a regularization term to help learn such a personalized weighted knowledge graph.

3.3.2 KGAT [35]. KGAT shares a generally similar idea with KGCN. The main difference lies in an auxiliary loss for knowledge graph reconstruction and the pretrained BPR-MF [27] features as inputs. Although not detailed in its paper, an important contribution of KGAT is to introduce the pretrained features into this tasks, which greatly improves the performance. Based on this finding, we successfully simplify KGAT and obtain similar or even better performance (See Table 5, denoted as KGAT–).

3.4 Summary

In summary, the prime common issue of existing HGNNs is the lack of fair comparison with homogeneous GNNs and other works—to some extent—encourage the new models to equip themselves with novel yet redundant modules, instead of focusing more on progress in performance. Additionally, a non-negligible proportion of works have individual issues, e.g., data leakage [12], tuning on test set [45], and two-order-of-magnitude more memory and time consumption without effectiveness improvements [43].

In light of the significant discrepancy, we take the initiative to setup a heterogeneous graph benchmark (HGB) with these three tasks on diverse datasets for open, reproducible heterogeneous graph research (See §4). Inspired by the promising advantages of the simple GAT over dedicated and relatively-complex heterogeneous GNN models, we present a simple heterogeneous GNN model with GAT as backbone, offering promising results on HGB (See §5).

4 HETEROGENEOUS GRAPH BENCHMARK

4.1 Motivation and Overview

Issues with current datasets. Several types of datasets—academic networks (e.g., ACM, DBLP), information networks (e.g., IMDB, Reddit), and recommendation graphs (e.g., Amazon, MovieLens)—are the most frequently-used datasets, but the detailed task settings could be quite different in different papers. For instance, HAN [36] and GTN [43] discard the citation links in ACM, while others use the original version. Besides, different splits of the dataset also contribute to uncomparable results. Finally, the recent graph benchmark OGB [18] mostly focuses on benchmarking graph machine learning methods on homogeneous graphs and is not dedicated to heterogeneous graphs.

Issues with current pipelines. To fulfill a task, components outside HGNNs can also play critical roles. For example, MAGNN [12] finds that not all types of node features are useful, and a pre-selection based on validation set could be helpful (See §4.3). RGCN [28] uses DistMult [39] instead of dot product for training in link prediction. We need to control the other components in the pipeline for fair comparison.

HGB. In view of these practical issues, we present the heterogeneous graph benchmark (HGB) for open, reproducible heterogeneous GNN research. We standardize the process of data splits, feature processing, and performance evaluation, by establishing the HGB pipeline “feature preprocessing \(\rightarrow\) HGNN encoder \(\rightarrow\) downstream decoder”. For each model, HGB selects the best fit feature preprocessing and downstream decoder based on its performance on validation set.

4.2 Dataset Construction

HGB collects 11 widely-recognized medium-scale datasets with predefined meta-paths from previous works, making it available to all kinds of HGNNs. The statistics are summarized in Table 2.

4.2.1 Node Classification. Node Classification follows a transductive setting, where all edges are available during training and node labels are split according to 24% for training, 6% for validation and 70% for test in each dataset.

- **DBLP** is a bibliography website of computer science. We use a commonly used subset in 4 areas with nodes representing authors, papers, terms and venues.
- **IMDB** is a website about movies and related information. A subset from Action, Comedy, Drama, Romance and Thriller classes is used.
- **ACM** is also a citation network. We use the subset hosted in HAN [36], but preserve all edges including paper citations and references.

---

[^2]: http://web.cs.ucla.edu/~yzsun/data/
[^3]: https://www.kaggle.com/karrrimba/movie-metadatacsv
• Freebase [3] is a huge knowledge graph. We sample a subgraph of 8 genres of entities with about 1,000,000 edges following the procedure of a previous survey [41].

4.2.2 Link Prediction. Link prediction is formulated as a binary classification problem in HGB. The edges are split according to 81% for training, 9% for validation and 10% for test. Then the graph is reconstructed only by edges in the training set. For negative node pairs in testing, we firstly tried uniform sampling and found that most models could easily make a nearly perfect prediction (See Appendix B). Finally, we sample 2-hop neighbors for negative node pairs, of which are 1:1 ratio to the positive pairs in the test set.

• Amazon is an online purchasing platform. We use the subset preprocessed by GATNE [5], containing electronics category products with co-viewing and co-purchasing links between them.

• LastFM is an online music website. We use the subset released by HetRec 2011 [4], and preprocess the dataset by filtering out the users and tags with only one link.

• PubMed is a biomedical literature library. We use the subset constructed by HNE [41].

4.2.3 Knowledge-aware recommendation. We randomly split 20% of user-item interactions as test set for each user, and for the left 80% interactions as training set.

• Amazon-book is a subset of Amazon-review related to books.

• LastFM is a subset extracted from last.fm with timestamp from January, 2015 to June, 2015.

• Yelp-2018 is a dataset adapted from 2018 edition of the Yelp challenge. Local businesses like restaurants and bars are seen as items.

• Movielens is a subset of Movielens-20M, which is a widely used dataset for recommendation. To assure the quality of dataset, we use 10-core setting to filter low-frequency nodes. To align items to knowledge graph entities, we adopt the same procedure as [34, 35].

4.3 Feature Preprocessing
As pointed out in § 4.1, the preprocessing for input features has a great impact on the performance. Our preprocessing methods are as follows.

Linear Transformation. As the input feature of different types of nodes may vary in dimension, we use a linear layer with bias for each node type to map all node features to a shared feature space. The parameters in these linear layers will be optimized along with the following HGNN.

Useful Types Selection. In many datasets, only features of a part of types are useful to the task. We can select a subset of node types to keep their features, and replace the features of nodes of other types as one-hot vectors. Combined with linear transformation, the replacement is equivalent to learn an independent embedding for each node of the unselected types. Ideally, we should enumerate all subsets of types and report the best one based on the performance on the validation set, but due to the high consumption to train the model $2^{\mathcal{E}}$ times, we decide to only enumerate three choices, i.e., using all given node features, using only features of target node type, or replacing all node features as one-hot vectors.

4.4 Downstream Decoders and Loss function
4.4.1 Node Classification. After setting the final dimension of HGNNs the same as the number of classes, we then adopt the most usual loss functions. For single-label classification, we use softmax and cross-entropy loss. For multi-label datasets, i.e. IMDB in HGB, we use a sigmoid activation and binary cross-entropy loss. For multi-label datasets, i.e. IMDB in HGB, we use a sigmoid activation and binary cross-entropy loss. For multi-label datasets, i.e. IMDB in HGB, we use a sigmoid activation and binary cross-entropy loss.

4.4.2 Link Prediction. As RGCN [2] suggests, DistMult [39] performs better than direct dot product, due to multiple types of edges, i.e. for node pair $u, v$ and a target edge type $r$,

$$\text{Prob}_r(u, v \text{ is positive}) = \text{sigmoid}[\text{HGNN}(u)^T R_r \text{HGNN}(v)],$$

where $R_r$ is a learnable square matrix (sometimes regularized with diagonal matrix) for type $r \in \mathcal{T}_e$. We find that DistMult outperforms dot product sometimes even when there is only single type of edge to predict. We try both dot product and DistMult decoders, and report the best results. The loss function is binary cross-entropy.

4.4.3 Knowledge-aware Recommendation. Recommendation is similar to link prediction, but differs in data distribution and focuses more on ranking. We define the similarity function $f(u, v)$ between nodes $u, v$ based on dot product. As mentioned in § 3.3.2, pretrained BPR-MF embeddings are of vital importance. We incorporate the BPR-MF embeddings $e_u, e_v$ via a bias term in $f(u, v)$ to avoid modification on the input or architectures of other models, i.e.,

$$f(u, v) = \text{HGNN}(u)^T \text{HGNN}(v) + e_u^T e_v.$$  

Following KGAT [35], we opt for BPR [27] loss for training.

$$\text{Loss}(u, v^+, v^-) = -\log \text{sigmoid}(f(u, v^+) - f(u, v^-)),$$  

### Table 2: Statistics of HGB datasets.

<table>
<thead>
<tr>
<th>Type</th>
<th>#Nodes</th>
<th>#Node Types</th>
<th>#Edges</th>
<th>#Edge Types</th>
<th>Target</th>
<th>#Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>26,128</td>
<td>4</td>
<td>239,566</td>
<td>6</td>
<td>author</td>
<td>4</td>
</tr>
<tr>
<td>IMDB</td>
<td>21,420</td>
<td>4</td>
<td>86,642</td>
<td>6</td>
<td>movie</td>
<td>5</td>
</tr>
<tr>
<td>ACM</td>
<td>10,942</td>
<td>4</td>
<td>547,872</td>
<td>8</td>
<td>paper</td>
<td>3</td>
</tr>
<tr>
<td>Freebase</td>
<td>180,098</td>
<td>8</td>
<td>1,057,688</td>
<td>36</td>
<td>book</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>#Users</th>
<th>#Items</th>
<th>#Interactions</th>
<th>#Entities</th>
<th>#Relations</th>
<th>#Triplets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon-book</td>
<td>70,679</td>
<td>23,566</td>
<td>37,385</td>
<td>45,919</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>LastFM</td>
<td>24,915</td>
<td>48,123</td>
<td>6,182</td>
<td>136,499</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Movielens</td>
<td>10,942</td>
<td>547,872</td>
<td>8</td>
<td>21,420</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Yelp-2018</td>
<td>10,612</td>
<td>4</td>
<td>141,521</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>
where \((u, v^+\)\) is a positive pair, and \((u, v^-)\) is a negative pair sampled at random.

### 4.5 Evaluation Settings

We evaluate all methods for all datasets by running 5 times with different random seeds, and reporting the average score and standard deviation.

#### 4.5.1 Node Classification

We evaluate node classification with Macro-F1 and Micro-F1 metrics for both multi-class (DBLP, ACM, Freebase) and multi-label (IMDB) datasets. The implementation is based on scikit-learn.

#### 4.5.2 Link Prediction

We evaluate link prediction with ROC-AUC (area under the ROC curve) and MRR (mean reciprocal rank) metrics.

Though GAT has powerful capacity in modeling homogeneous graphs, it may be not optimal for heterogeneous graphs due to the neglect of node or edge types. To tackle this problem, we extend GAT as backbone with enhancements from the redesign of three well-known techniques: Learnable edge-type embedding, residual connections, and L2 normalization on the output embeddings. Figure 1 illustrates the full pipeline with Simple-HGN.

#### 5 A SIMPLE HETEROGENEOUS GNN

Inspired by the advantage of the simple GAT over advanced and dedicated heterogeneous GNNs, we present Simple-HGN, a simple and effective method for modeling heterogeneous graph. Simple-HGN adopts GAT as backbone with enhancements from the redesign of three well-known techniques: Learnable edge-type embedding, residual connections, and L2 normalization on the output embeddings. Figure 1 illustrates the full pipeline with Simple-HGN.

#### 5.1 Learnable Edge-type Embedding

Though GAT has powerful capacity in modeling homogeneous graphs, it may be not optimal for heterogeneous graphs due to the neglect of node or edge types. To tackle this problem, we extend the original graph attention mechanism by including edge type information into attention calculation. Specifically, at each layer, we allocate a \(d_l\)-dimensional embedding \(\hat{\psi}(e)\) for each edge type \(e \in T_e\), and use both edge type embeddings and node embeddings to calculate the attention score as follows:

\[
\hat{a}_{ij} = \frac{\exp\{\text{LeakyReLU}\left(a^T[W_h_i][W_h_j][W_r \hat{\psi}(i,j)]\right)\}}{\sum_{k \in N_i} \exp\{\text{LeakyReLU}\left(a^T[W_h_i][W_h_k][W_r \hat{\psi}(i,k)]\right)\}},
\]

where \(\hat{\psi}(i,j)\) represents the type of edge between node \(i\) and node \(j\), and \(W_r\) is a learnable matrix to transform type embeddings.

#### 5.2 Residual Connection

GNNs are hard to be deep due to the over-smoothing and gradient vanishing problems [23, 38]. A famous solution to mitigate this problem in computer vision is residual connection [15]. However, the original GCN paper [21] showed a negative result for residual connection on graph convolution. Recent study [22] finds that well-designed pre-activation implementation could make residual connection great again in GNNs.

**Node Residual.** We add pre-activation residual connection for node representation across layers. The aggregation at the \(l\)th layer can be expressed as

\[
h^{(l)}_i = \sigma \left( \sum_{j \in N_i} a^{(l)}_{ij} W^{(l)} h^{(l-1)}_j + h^{(l-1)}_i \right),
\]

where \(a^{(l)}_{ij}\) is the attention weight about edge \((i, j)\) and \(\sigma\) is an activation function (ELU [6] by default). When the dimension changes in the \(l\)-th layer, an additional learnable linear transformation \(W^{(l)} \in \mathbb{R}^{d_{in} \times d_{out}}\) is needed, i.e.,

\[
h^{(l)}_i = \sigma \left( \sum_{j \in N_i} a^{(l)}_{ij} W^{(l)} h^{(l-1)}_j + W^{(l)}_{res} h^{(l-1)}_i \right).
\]

**Edge Residual.** Recently, Realformer [16] reveals that residual connection on attention scores is also helpful. After getting the raw attention scores \(\hat{a}\) via Eq. (7), we add residual connections to them,

\[
a^{(l)}_{ijk} = (1 - \beta) a^{(l)}_{ijk} + \beta a^{(l-1)}_{ijk},
\]

where hyperparameter \(\beta \in [0, 1]\) is a scaling factor.

**Multi-head Attention.** Similar to GAT, we adopt multi-head attention to enhance model’s expressive capacity. Specifically, we perform \(K\) independent attention mechanisms according to Equation (8), and concatenate their results as the final representation. The corresponding updating rule is:

\[
a^{(l)}_{ijk} = (1 - \beta) a^{(l)}_{ijk} + \beta a^{(l-1)}_{ijk},
\]

\[
\hat{h}^{(l)}_{ik} = \sum_{j \in N_i} a^{(l)}_{ijk} W^{(l)}_{k} h^{(l-1)}_j,
\]

\[
h^{(l)}_i = \sigma \left( \frac{K}{\sum_{k=1}^{K} \hat{h}^{(l)}_{ik} + W^{(l)}_{res} h^{(l-1)}_i} \right).
\]

where \(\|\) denotes concatenation operation, and \(\hat{a}^{(l)}_{ijk}\) is attention score computed by the \(k\)th linear transformation \(W^{(l)}_k\) according to Equation (9).

Usually the output dimension cannot be divided exactly by the number of heads. Following GAT, we no longer use concatenation but adopt averaging for the representation in the final \((Lth)\) layer, i.e.,

\[
h^{(L)}_i = \frac{1}{K} \sum_{k=1}^{K} \hat{h}^{(L)}_{ik}.
\]

**Adaptation for Link Prediction.** We slightly modify the model architecture for better performance on link prediction. Edge residual is removed and the final embedding is the concatenation of
embeddings from all the layers. This adapted version is similar to JKNet [38].

5.3 $L_2$ Normalization

We find that an $L_2$ normalization on the output embedding is extremely useful, i.e.,

$$o_i = \frac{h_i^{(L)}}{\|h_i^{(L)}\|},$$  \hspace{2cm} (15)
We benchmark results for 1) all HGNNs discussed in Section 3, 2) where KGAT- refers to KGAT without redundant designs (See § 3.3.2). GCN and GAT are not included, because they are already very similar to KGCN and KGAT-. The KGCN and KGAT works focus more on incorporating knowledge into user-item graphs than new architectures.

Tables 3, 4, and 5 report results for node classification, link prediction, and knowledge-aware recommendation benchmark. KGAT- refers to KGAT without redundant designs (See § 3.3.2). The only modification occurs on their data loading interfaces and downstream decoders, if necessary, to make their codes adapt to the HGB pipeline.

Implementations of all previous HGNNs are based on their official codes to avoid errors introduced by re-implementation. The only modification occurs on their data loading interfaces and downstream decoders, if necessary, to make their codes adapt to the HGB pipeline.

We use Adam optimizer with weight decay for all methods, and tune hyperparameters based on the validation set performance. The details of hyperparameters are recorded in Appendix C. For the methods requiring meta-paths, the meta-paths used in benchmark datasets are shown in Appendix D.

### 6.1 Benchmark

Tables 3, 4, and 5 report results for node classification, link prediction, and knowledge-aware recommendation, respectively. The results show that under fair comparison, 1) the simple homogeneous GAT can matches the best HGNNs in most cases, and 2) inherited from GAT, Simple-HGN consistently outperforms all advanced HGNNs methods for node classification on four datasets, link prediction on three datasets, and knowledge-aware recommendation on three datasets.

Implementations of all previous HGNNs are based on their official codes to avoid errors introduced by re-implementation. The only modification occurs on their data loading interfaces and downstream decoders, if necessary, to make their codes adapt to the HGB pipeline.

We use Adam optimizer with weight decay for all methods, and tune hyperparameters based on the validation set performance. The details of hyperparameters are recorded in Appendix C. For the methods requiring meta-paths, the meta-paths used in benchmark datasets are shown in Appendix D.

### 6 EXPERIMENTS

We benchmark results for 1) all HGNNs discussed in Section 3, 2) GCN and GAT, and 3) Simple-HGN on HGB. All experiments are reported with the average and the standard variance of five runs.

### 6.2 Time and Memory Consumption

We test the time and memory consumption of all available HGNNs for node classification on the DBLP dataset. The results are showed in Figure 2. It is worth noting that we only measure the time consumption of one epoch for each model, but the needed number of epochs until convergence could be various and hard to exactly define. HetSANN is omitted due to our failure to get a reasonable Micro-F1 score.

### 6.3 Ablation Studies

The ablation studies on all the three tasks are summarized in Table 6. Residual connection and $L_2$ normalization consistently improve performance, but type embedding only slightly boosts the performance on node classification, although it is the best way in our experiments to encode type information explicitly under the GAT framework. We will discuss the possible reasons in § 7.

### 7 DISCUSSION AND CONCLUSION

In this work, we identify the neglected issues in heterogeneous GNNs, setup the heterogeneous graph benchmark (HGB), and introduce a simple and strong baseline Simple-HGN. The goal of this work is to understand and advance the developments of heterogeneous GNNs by facilitating reproducible and robust research. Notwithstanding the extensive and promising results, there are still open questions remaining for heterogeneous GNNs and broadly heterogeneous graph representation learning.

Is explicit type information useful? Ablation studies in Table 6 suggest the type embeddings only bring minor improvements. We hypothesize that the main reason is that the heterogeneity of node features already implies the different node and edge types. Another
possibility is that the current graph attention mechanism [32] is too weak to fuse the type information with feature information. We leave this question for future study.

Are meta-paths or variants still useful in GNNs? Meta-paths [29] are proposed to separate different semantics with human prior. However, the premise of (graph) neural networks is to avoid the feature engineering process by extracting implicit and useful features underlying the data. Results in previous sections also suggest that meta-path based GNNs do not generate outperformance over the homogeneous GAT. Are there better ways to leverage meta-paths in heterogeneous GNNs than existing attempts? Will meta-paths still be necessary for heterogeneous GNNs in the future and what are the substitutions?

ACKNOWLEDGMENTS

The work is supported by the NSFC for Distinguished Young Scholar (61825602) and NSFC (61836103). The authors would like to thank Haonan Wang from UIUC and Hongxia Yang from Alibaba for their kind feedbacks.

REFERENCES

[22] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. PVLDB 4, 11 (2011), 992–1003.
[27] Huiting Hong, Hantao Guo, Yuancheng Lin, Xiaoying Yang, Zang Li, and Jieping Ye. 2020. An attention-based graph neural network for heterogeneous structural
A TIME AND MEMORY CONSUMPTION

![Figure 2: Time and memory comparison for HGNNs on DBLP. The area of the circles represent the (relative) memory consumption of the corresponding models.](image)

B RANDOM NEGATIVE

The distribution of negative samples of test set in link prediction task has a great impact on the performance score. The results with random negative test in our benchmark are shown in Table 7. As we can see, the scores are greater than those in Table 4 by a large margin. Most works [5, 12, 44] evaluate link prediction with randomly sampled negative pairs, which are easy to distinguish from the positive pairs for most methods. However, in real world scenarios, we usually need to discriminate positive and negative node pairs with similar characters, instead of random ones, due to the widely used “retrieve then re-rank” industrial pipeline. Therefore, we choose to use sampled 2-hop neighbors as our negative test set in benchmark.

C HYPER-PARAMETERS

We search learning rate within $[1, 5] \times \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ in all cases, and $[0, 1, 2, 5] \times \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ for weight decay rate. We set dropout rate as 0.1 in recommendation, and 0.5 in node classification and link prediction by default. For batch size, we will try $[1024, 2048, 4096, 8192]$, unless the code of the author has special requirements. For training epoch, we will use early stop mechanism based on the evaluation on validation set to promise fully training.

For brevity, we will denote some variables. Suppose dimension of embeddings for graph layers as $d$, dimension of edge embeddings as $d_{e}$, dimension of attention vector (if exists) as $d_{a}$, number of graph layers as $L$, number of attention heads as $n_{h}$, negative slope of LeakyReLU as $s$.

For input feature type, we use feat $= 0$ to denote using all given features, feat $= 1$ to denote using only target node features, and feat $= 2$ to denote all nodes with one-hot features.

C.1 Simple-HGN

C.1.1 Node classification. We set $d_{e} = 64$, $n_{h} = 8$, $\beta = 0.05$ for all datasets. For DBLP, ACM and Freebase datasets, we set $L = 3$, $s = 0.05$. For IMDB dataset, we set $L = 6$, $s = 0.1$. We set $\text{feat} = 0$ for IMDB, $\text{feat} = 1$ for ACM, and $\text{feat} = 2$ for DBLP and Freebase.

C.1.2 Link prediction. We set $d_{e} = 64$, $d_{a} = 32$, $n_{h} = 2$, $\beta = 0$, $s = 0.01$ for all datasets. For Amazon and PubMed, we use DistMult as decoder, and set $L = 3$. For LastFM, we use dot product as decoder, and set $L = 4$. We use feat $= 2$ for all datasets.

C.1.3 Recommendation. For all datasets, we set $d^{(0)} = 64$, $d^{(1)} = 32$, $d^{(2)} = 16$, $n_{h} = 1$, $s = 0.01$ as suggested in [35].

C.2 HAN

C.2.1 Node classification. We set $d = 8$, $d_{a} = 128$, $n_{h} = 8$ and $L = 2$ for all datasets. For input feature type, we use feat $= 2$ in Freebase, and feat $= 1$ in other datasets. We have also tried larger $d$, but the variation of performance becomes very large. Therefore, we keep $d = 8$ as suggested in HAN’s code.

C.3 GTN

C.3.1 Node classification. We use adaptive learning rate suggested in their paper for all datasets. We set $d = 64$, number of GTN channels as 2. For DBLP and ACM, we set $L = 2$. For IMDB dataset, we set $L = 3$.

Moreover, as suggested in GTN paper, we aggregate the keyword node information as attribute to neighbors and use the left subgraph to do node classification. We also tried to use the whole graph for GTN. Unfortunately, it collapse in that case, which indicates GTN is sensitive to the graph structure.

C.4 RSHN

For IMDB and DBLP, we set $L = 2$ and $d = 16$. For ACM, we set $L = 3$ and $d = 32$. We use feat $= 0, 1, 2$ for ACM, IMDB and DBLP respectively.

C.5 HetGNN

C.5.1 Node classification. We set $d = 128$, feat $= 0$, and batch size as 200 for all datasets. For random walk, we set walk length as 30 and the window size as 5.

C.5.2 Link prediction. We set $d = 128$, feat $= 2$, and batch size as 200 for all datasets. For random walk, we set walk length as 30 and the window size as 5.

C.6 MAGNN

We set $d = 64$, $d_{a} = 128$ and $n_{h} = 8$ in all cases.

C.6.1 Node classification. We use feat $= 1$ in all cases. For DBLP and ACM datasets, we set batch size as 8, and number of neighbor samples as 100. For IMDB dataset, we use full batch training.

C.6.2 Link prediction. We set batch size as 8, and number of neighbor samples as 100 for LastFM. For other datasets, we failed to adapt the MAGNN code to them because there is too much hard-coding.

C.7 HetSANN

C.7.1 Node classification. For ACM, we set $d = 64$, $L = 3$, $n_{h} = 8$ and feat $= 0$. For IMDB, we set $d = 32$, $L = 2$, $n_{h} = 4$ and feat $= 1$. For DBLP, we set $d = 64$, $L = 2$, $n_{h} = 4$ and feat $= 2$.

C.8 HGT

C.8.1 Node Classification. We use layer normalization in each layer, and set $d = 64$, $n_{h} = 8$ for all datasets. $L$ is set to 2, 3, 3, 5
for ACM, DBLP, Freebase and IMDB respectively. For input feature type, we use feat = 2 in Freebase, feat = 1 in IMDB and DBLP, and feat = 0 in ACM.

C.8.2 Link Prediction. For all datasets, we use layer normalization in each layer, and set \( d = 64 \), \( n_h = 8 \), feat = 2 and DistMult as decoder.

C.9 GCN

C.9.1 Node classification. We set \( d = 64 \) for all datasets. We set \( L = 3 \) for DBLP, ACM and Freebase, and \( L = 4 \) for IMDB. We use feat = 2 for DBLP and Freebase, and feat = 0 for ACM and IMDB.

C.9.2 Link prediction. We set \( d = 64 \), \( L = 2 \), and feat = 2 for all datasets.

C.10 GAT

C.10.1 Node classification. We set \( d = 64 \), \( n_h = 8 \) for all datasets. For DBLP, ACM and Freebase, we set \( s = 0.05 \) and \( L = 3 \). For IMDB, we set \( s = 0.1 \) and \( L = 5 \). We use feat = 2 for DBLP and Freebase, feat = 1 for ACM, and feat = 0 for IMDB.

C.10.2 Link prediction. We set \( d = 64 \), \( n_h = 4 \), \( L = 3 \) and feat = 2 for all datasets.

C.11 RGCN

C.11.1 Node classification. We set \( L = 5 \) for all datasets. For ACM, we set \( d = 16 \), feats = 2. For DBLP and Freebase, we set \( d = 16 \), feats = 3. For IMDB, we set \( d = 32 \), feats = 1.

C.11.2 Link prediction. We set \( L = 4 \), \( d = 64 \) and feat = 2 for all datasets.

C.12 GATNE

C.12.1 Link prediction. We set \( d = 200 \), \( d_c = 10 \), \( d_o = 20 \), feat = 2 for all datasets.

For random walk, we set walk length as 30 and the window size as 5. For neighbor sampling, we set negative samples for optimization as 5, neighbor samples for aggregation as 10.

C.13 KGCN and KGNN-LS

C.13.1 Recommendation. For all datasets, we set \( d^{(0)} = 64 \) and \( d^{(1)} = 48 \). We also tried to stack more graph layers, but performance deteriorates when we do that, which is also found in [33, 34].