Heterogeneous Graph Transformer

Ziniu Hu*
University of California, Los Angeles
bull@cs.ucla.edu

Yuxiao Dong
Microsoft Research, Redmond
yuxdong@microsoft.com

Kuansan Wang
Microsoft Research, Redmond
kuansanw@microsoft.com

Yizhou Sun
University of California, Los Angeles
yzsun@cs.ucla.edu

ABSTRACT

Recent years have witnessed the emerging success of graph neural networks (GNNs) for modeling structured data. However, most GNNs are designed for homogeneous graphs, in which all nodes and edges belong to the same types, making them infeasible to represent heterogeneous structures. In this paper, we present the Heterogeneous Graph Transformer (HGT) architecture for modeling Web-scale heterogeneous graphs. To model heterogeneity, we design node- and edge-type dependent parameters to characterize the heterogeneous attention over each edge, empowering HGT to maintain dedicated representations for different types of nodes and edges. To handle dynamic heterogeneous graphs, we introduce the relative temporal encoding technique into HGT, which is able to capture the dynamic structural dependency with arbitrary durations. To handle Web-scale graph data, we design the heterogeneous mini-batch graph sampling algorithm—HGSampling—for efficient and scalable training. Extensive experiments on the Open Academic Graph of 179 million nodes and 2 billion edges show that the proposed HGT model consistently outperforms all the state-of-the-art GNN baselines by 9%–21% on various downstream tasks. The dataset and source code of HGT are publicly available at https://github.com/acbull/pyHGT.

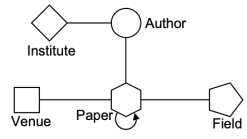
KEYWORDS

Graph Neural Networks; Heterogeneous Information Networks; Representation Learning; Graph Embedding; Graph Attention

ACM Reference Format:
Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380027

1 INTRODUCTION

Heterogeneous graphs have been commonly used for abstracting and modeling complex systems, in which objects of different types interact with each other in various ways. Some prevalent instances of such systems include academic graphs, Facebook entity graph, LinkedIn economic graph, and broadly the Internet of Things network. For example, the Open Academic Graph (OAG) [28] in Figure 1 contains five types of nodes: papers, authors, institutions, venues (journal, conference, or preprint), and fields, as well as different types of relationships between them.



(a) The schema of heterogeneous academic networks

Author	*is_(first/last/other)_author_of*	Paper
Author	*is_affiliated_with*	Institute
Paper	*is_published_(conf/journal)_at*	Venue
Paper	*has_(L_1–L_5)_field_of*	Field
Paper	*has_citation_to*	Paper

(b) The meta relations of heterogeneous academic networks

Figure 1: The schema and meta relations of Open Academic Graph (OAG). Given a Web-scale heterogeneous graph, e.g., an academic network, HGT takes only its one-hop edges as input without manually designing meta paths.

Over the past decade, a significant line of research has been explored for mining heterogeneous graphs [17]. One of the classical paradigms is to define and use meta paths to model heterogeneous structures, such as PathSim [18] and metapath2vec [3]. Recently, in view of graph neural networks' (GNNs) success [7, 9, 22], there are several attempts to adopt GNNs to learn with heterogeneous networks [14, 23, 26, 27]. However, these works face several issues: First, most of them involve the design of meta paths for each type of heterogeneous graphs, requiring specific domain knowledge; Second, they either simply assume that different types of nodes/edges share the same feature and representation space or keep distinct non-sharing weights for either node type or edge type alone, making them insufficient to capture heterogeneous graphs' properties; Third, most of them ignore the dynamic nature of every (heterogeneous) graph; Finally, their intrinsic design and implementation make them incapable of modeling Web-scale heterogeneous graphs.

Take OAG for example: First, the nodes and edges in OAG could have different feature distributions, e.g., papers have text features whereas institutions may have features from affiliated scholars, and coauthorships obviously differ from citation links; Second, OAG has been consistently evolving, e.g., 1) the volume of publications doubles every 12 years [4], and 2) the KDD conference was more related to database in the 1990s whereas more to machine learning in recent years; Finally, OAG contains hundreds of millions of nodes

*This work was done when Ziniu was an intern at Microsoft Research.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380027

and billions of relationships, leaving existing heterogeneous GNNs not scalable for handling it.

In light of these limitations and challenges, we propose to study heterogeneous graph neural networks with the goal of maintaining node- and edge-type dependent representations, capturing network dynamics, avoiding customized meta paths, and being scalable to Web-scale graphs. In this work, we present the Heterogeneous Graph Transformer (HGT) architecture to deal with all these issues.

To handle graph heterogeneity, we introduce the node- and edge-type dependent attention mechanism. Instead of parameterizing each type of edges, the heterogeneous mutual attention in HGT is defined by breaking down each edge $e = (s, t)$ based on its meta relation triplet, i.e., $\langle$ node type of s, edge type of e between s & t, node type of $t\rangle$. Figure 1 illustrates the meta relations of heterogeneous academic graphs. In specific, we use these meta relations to parameterize the weight matrices for calculating attention over each edge. As a result, nodes and edges of different types are allowed to maintain their specific representation spaces. Meanwhile, connected nodes in different types can still interact, pass, and aggregate messages without being restricted by their distribution gaps. Due to the nature of its architecture, HGT can incorporate information from high-order neighbors of different types through message passing across layers, which can be regarded as "soft" meta paths. That said, even if HGT take only its one-hop edges as input without manually designing meta paths, the proposed attention mechanism can automatically and implicitly learn and extract "meta paths" that are important for different downstream tasks.

To handle graph dynamics, we enhance HGT by proposing the relative temporal encoding (RTE) strategy. Instead of slicing the input graph into different timestamps, we propose to maintain all the edges happening in different times as a whole, and design the RTE strategy to model structural temporal dependencies with any duration length, and even with unseen and future timestamps. By end-to-end training, RTE enables HGT to automatically learn the temporal dependency and evolution of heterogeneous graphs.

To handle Web-scale graph data, we design the first heterogeneous sub-graph sampling algorithm—HGSampling—for mini-batch GNN training. Its main idea is to sample heterogeneous sub-graphs in which different types of nodes are with similar proportions, since the direct usage of existing (homogeneous) GNN sampling methods, such as GraphSage [7], FastGCN [1], and LADIES [29], results in highly imbalanced ones regarding to both node and edge types. In addition, it is also designed to keep the sampled sub-graphs dense for minimizing the loss of information. With HGSampling, all the GNN models, including our proposed HGT, can train and infer on arbitrary-size heterogeneous graphs.

We demonstrate the effectiveness and efficiency of the proposed Heterogeneous Graph Transformer on the Web-scale Open Academic Graph comprised of 179 million nodes and 2 billion edges spanning from 1900 to 2019, making this the largest-scale and longest-spanning representation learning yet performed on heterogeneous graphs. Additionally, we also examine it on domain-specific graphs: the computer science and medicine academic graphs. Experimental results suggest that HGT can significantly improve various downstream tasks over state-of-the-art GNNs as well as dedicated heterogeneous models by 9–21%. We further conduct case studies to show the proposed method can indeed automatically capture the importance of implicit meta paths for different tasks.

2 PRELIMINARIES AND RELATED WORK

In this section, we introduce the basic definition of heterogeneous graphs with network dynamics and review the recent development on graph neural networks (GNNs) and their heterogeneous variants. We also highlight the difference between HGT and existing attempts on heterogeneous graph neural networks.

2.1 Heterogeneous Graph Mining

Heterogeneous graphs [17] (a.k.a., heterogeneous information networks) are an important abstraction for modeling relational data for many real-world complex systems. Formally, it is defined as:

Definition 1. Heterogeneous Graph: A heterogeneous graph is defined as a directed graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R})$ where each node $v \in \mathcal{V}$ and each edge $e \in \mathcal{E}$ are associated with their type mapping functions $\tau(v) : V \rightarrow \mathcal{A}$ and $\phi(e) : E \rightarrow \mathcal{R}$, respectively.

Meta Relation. For an edge $e = (s, t)$ linked from source node s to target node t, its meta relation is denoted as $\langle \tau(s), \phi(e), \tau(t)\rangle$. Naturally, $\phi(e)^{-1}$ represents the inverse of $\phi(e)$. The classical meta path paradigm [17–19] is defined as a sequence of such meta relation.

Notice that, to better model real-world heterogeneous networks, we assume that there may exist multiple types of relations between different types of nodes. For example, in OAG there are different types of relations between the *author* and *paper* nodes by considering the authorship order, i.e., "the first author of", "the second author of", and so on.

Dynamic Heterogeneous Graph. To model the dynamic nature of real-world (heterogeneous) graphs, we assign an edge $e = (s, t)$ a timestamp T, when node s connects to node t at T. If s appears for the first time, T is also assigned to s. s can be associated with multiple timestamps if it builds connections over time.

In other words, we assume that the timestamp of an edge is unchanged, denoting the time it is created. For example, when a paper published on a conference at time T, T will be assigned to the edge between the paper and conference nodes. On the contrary, different timestamps can be assigned to a node accordingly. For example, the *conference* node "WWW" can be assigned any year. WWW@1994 means that we are considering the first edition of WWW, which focuses more on internet protocol and Web infrastructure, while WWW@2020 means the upcoming WWW, which expands its research topics to social analysis, ubiquitous computing, search & IR, privacy and society, etc.

There have been significant lines of research on mining heterogenous graphs, such as node classification, clustering, ranking and representation learning [3, 17–19], while the dynamic perspective of HGs has not been extensively explored and studied.

2.2 Graph Neural Networks

Recent years have witnessed the success of graph neural networks for relational data [7, 9, 22]. Generally, a GNN can be regarded as using the input graph structure as the computation graph for

message passing [6], during which the local neighborhood information is aggregated to get a more contextual representation. Formally, it has the following form:

Definition 2. General GNN Framework: Suppose $H^l[t]$ is the node representation of node t at the (l)-th GNN layer, the update procedure from the $(l\text{-}1)$-th layer to the (l)-th layer is:

$$H^l[t] \leftarrow \underset{\forall s \in N(t), \forall e \in E(s,t)}{\textbf{Aggregate}} \left(\textbf{Extract}\left(H^{l-1}[s]; H^{l-1}[t], e \right) \right) \quad (1)$$

where $N(t)$ denotes all the source nodes of node t and $E(s,t)$ denotes all the edges from node s to t.

The most important GNN operators are Extract$(\cdot)$ and Aggregate$(\cdot)$. Extract$(\cdot)$ represents the neighbor information extractor. It extract useful information from source node's representation $H^{l-1}[s]$, with the target node's representation $H^{l-1}[t]$ and the edge e between the two nodes as query. Aggregate$(\cdot)$ gather the neighborhood information of souce nodes via some aggregation operators, such as *mean, sum,* and *max*, while more sophisticated pooling and normalization functions can be also designed.

Various (homogeneous) GNN architectures have been proposed following this framework. Kipf *et al.* [9] propose graph convolutional network (GCN), which averages the one-hop neighbor of each node in the graph, followed by a linear projection and non-linear activation operations. Hamilton *et al.* propose GraphSAGE that generalizes GCN's aggregation operation from *average* to *sum, max* and a *RNN unit*. Velickovi *et al.* propose graph attention network (GAT) [22] by introducing the attention mechanism into GNNs, which allows GAT to assign different importance to nodes within the same neighborhood.

2.3 Heterogeneous GNNs

Recently, studies have attempted to extend GNNs for modeling heterogeneous graphs. Schlichtkrull *et al.* [14] propose the relational graph convolutional networks (RGCN) to model knowledge graphs. RGCN keeps a distinct linear projection weight for each edge type. Zhang *et al.* [27] present the heterogeneous graph neural networks (HetGNN) that adopts different RNNs for different node types to integrate multi-modal features. Wang *et al.* [23] extend graph attention networks by maintaining different weights for different meta-path-defined edges. They also use high-level semantic attention to differentiate and aggregate information from different meta paths.

Though these methods have shown to be empirically better than the vanilla GCN and GAT models, they have not fully utilized the heterogeneous graphs' properties. All of them use either node type or edge type alone to determine GNN weight matrices. However, the node or edge counts of different types can vary greatly. For relations that don't have sufficient occurrences, it's hard to learn accurate relation-specific weights. To address this, we propose to consider parameter sharing for a better generalization. Given an edge $e = (s,t)$ with its meta relation as $\langle \tau(s), \phi(e), \tau(t) \rangle$, if we use three interaction matrices to model the three corresponding elements $\tau(s), \phi(e)$, and $\tau(t)$ in the meta relation, then the majority of weights could be shared. For example, in "the first author of" and "the second author of" relationships, their source and target node types are both *author* to *paper*, respectively. In other words, the knowledge about *author* and *paper* learned from one relation could be quickly transferred and adapted to the other one. Therefore, we integrate this idea with the powerful Transformer-like attention architecture, and propose Heterogeneous Graph Transformer.

To summarize, the key differences between HGT and existing attempts include:

(1) Instead of attending on node or edge type alone, we use the meta relation $\langle \tau(s), \phi(e), \tau(t) \rangle$ to decompose the interaction and transform matrices, enabling HGT to capture both the common and specific patterns of different relationships using equal or even fewer parameters.
(2) Different from most of the existing works that are based on customized meta paths, we rely on the nature of the neural architecture to incorporate high-order heterogeneous neighbor information, which automatically learns the importance of implicit meta paths.
(3) Most previous works don't take the dynamic nature of (heterogeneous) graphs into consideration, while we propose the relative temporal encoding technique to incorporate temporal information by using limited computational resources.
(4) None of the existing heterogeneous GNNs are designed for and experimented with Web-scale graphs, we therefore propose the heterogeneous Mini-Batch graph sampling algorithm designed for Web-scale graph training, enabling experiments on the billion-scale Open Academic Graph.

3 HETEROGENEOUS GRAPH TRANSFORMER

In this section, we present the Heterogeneous Graph Transformer (HGT). Its idea is to use the **meta relations** of heterogeneous graphs to parameterize weight matrices for the heterogeneous mutual attention, message passing, and propagation steps. To further incorporate network dynamics, we introduce a relative temporal encoding mechanism into the model.

3.1 Overall HGT Architecture

Figure 2 shows the overall architecture of Heterogeneous Graph Transformer. Given a sampled heterogeneous sub-graph (Cf. Section 4), HGT extracts all linked node pairs, where target node t is linked by source node s via edge e. The goal of HGT is to aggregate information from source nodes to get a contextualized representation for target node t. Such process can be decomposed into three components: *Heterogeneous Mutual Attention*, *Heterogeneous Message Passing* and *Target-Specific Aggregation*.

We denote the output of the (l)-th HGT layer as $H^{(l)}$, which is also the input of the $(l+1)$-th layer. By stacking L layers, we can get the node representations of the whole graph $H^{(L)}$, which can be used for end-to-end training or fed into downstream tasks.

3.2 Heterogeneous Mutual Attention

The first step is to calculate the mutual attention between source node s and target node t. We first give a brief introduction to the general attention-based GNNs as follows:

$$H^l[t] \leftarrow \underset{\forall s \in N(t), \forall e \in E(s,t)}{\textbf{Aggregate}} \left(\textbf{Attention}(s,t) \cdot \textbf{Message}(s) \right) \quad (2)$$

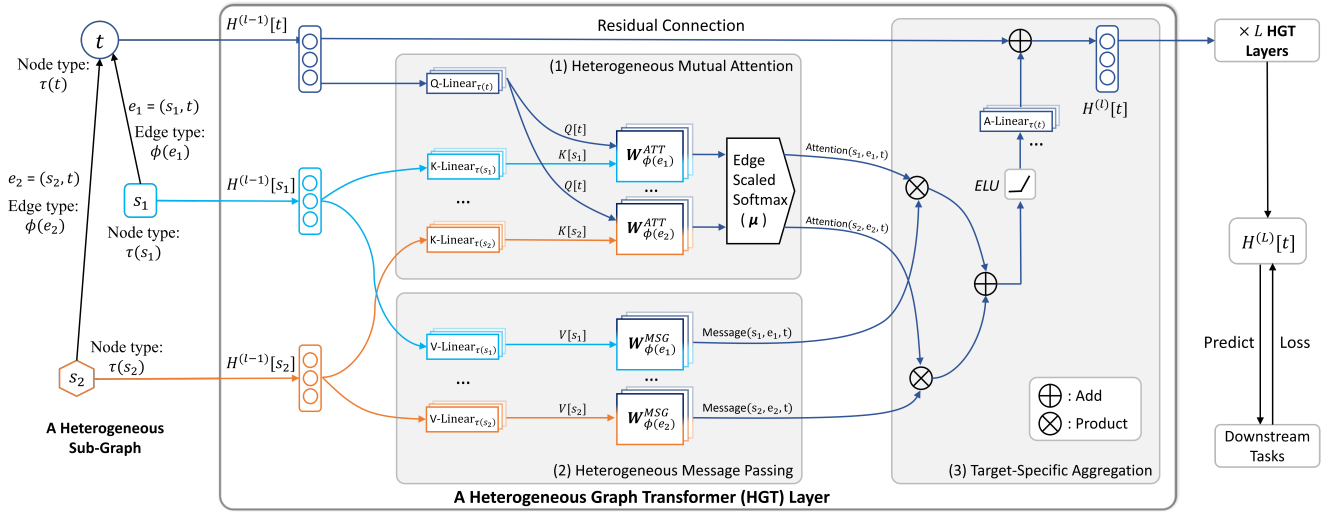


Figure 2: The Overall Architecture of Heterogeneous Graph Transformer. Given a sampled heterogeneous sub-graph with t as the target node, s_1 & s_2 as source nodes, the HGT model takes its edges $e_1 = (s_1, t)$ & $e_2 = (s_2, t)$ and their corresponding meta relations $< \tau(s_1), \phi(e_1), \tau(t) >$ & $< \tau(s_2), \phi(e_2), \tau(t) >$ as input to learn a contextualized representation $H^{(L)}$ for each node, which can be used for downstream tasks. Color decodes the node type. HGT includes three components: (1) meta relation-aware heterogeneous mutual attention, (2) heterogeneous message passing from source nodes, and (3) target-specific heterogeneous message aggregation.

where there are three basic operators: **Attention**, which estimates the importance of each source node; **Message**, which extracts the message by using only the source node s; and **Aggregate**, which aggregates the neighborhood message by the attention weight.

For example, the Graph Attention Network (GAT) [22] adopts an additive mechanism as **Attention**, uses the same weight for calculating **Message**, and leverages the simple average followed by a nonlinear activation for the **Aggregate** step. Formally, GAT has

$$\textbf{Attention}_{GAT}(s,t) = \underset{\forall s \in N(t)}{\text{Softmax}}\Big(\vec{a}\Big(WH^{l-1}[t] \parallel WH^{l-1}[s]\Big)\Big)$$
$$\textbf{Message}_{GAT}(s) = WH^{l-1}[s]$$
$$\textbf{Aggregate}_{GAT}(\cdot) = \sigma\Big(\text{Mean}(\cdot)\Big)$$

Though GAT is effective to give high attention values to important nodes, it assumes that s and t have the same feature distributions by using one weight matrix W. Such an assumption, as we've discussed in Section 1, is usually incorrect for heterogeneous graphs, where each type of nodes can have its own feature distribution.

In view of this limitation, we design the **Heterogeneous Mutual Attention** mechanism. Given a target node t, and all its neighbors $s \in N(t)$, which might belong to different distributions, we want to calculate their mutual attention grounded by their **meta relations**, i.e., the $\langle \tau(s), \phi(e), \tau(t) \rangle$ triplets.

Inspired by the architecture design of Transformer [21], we map target node t into a Query vector, and source node s into a Key vector, and calculate their dot product as attention. The key difference is that the vanilla Transformer uses a single set of projections for all words, while in our case each meta relation should have a distinct set of projection weights. To maximize parameter sharing while still maintaining the specific characteristics of different relations, we propose to parameterize the weight matrices of the interaction operators into a source node projection, an edge projection, and a target node projection. Specifically, we calculate the h-head attention for each edge $e = (s, t)$ (See Figure 2 (1)) by:

$$\textbf{Attention}_{HGT}(s,e,t) = \underset{\forall s \in N(t)}{\text{Softmax}}\Big(\Big\Vert_{i \in [1,h]} ATT\text{-}head^i(s,e,t)\Big) \quad (3)$$
$$ATT\text{-}head^i(s,e,t) = \Big(K^i(s)\, W^{ATT}_{\phi(e)}\, Q^i(t)^T\Big) \cdot \frac{\mu_{\langle \tau(s), \phi(e), \tau(t) \rangle}}{\sqrt{d}}$$
$$K^i(s) = \text{K-Linear}^i_{\tau(s)}\Big(H^{(l-1)}[s]\Big)$$
$$Q^i(t) = \text{Q-Linear}^i_{\tau(t)}\Big(H^{(l-1)}[t]\Big)$$

First, for the i-th attention head $ATT\text{-}head^i(s,e,t)$, we project the $\tau(s)$-type source node s into the i-th *Key* vector $K^i(s)$ with a linear projection $\text{K-Linear}^i_{\tau(s)} : \mathbb{R}^d \to \mathbb{R}^{\frac{d}{h}}$, where h is the number of attention heads and $\frac{d}{h}$ is the vector dimension per head. Note that $\text{K-Linear}^i_{\tau(s)}$ is indexed by the source node s's type $\tau(s)$, meaning that each type of nodes has a unique linear projection to maximally model the distribution differences. Similarly, we also project the target node t with a linear projection $\text{Q-Linear}^i_{\tau(t)}$ into the $i-$th Query vector.

Next, we need to calculate the similarity between the Query vector $Q^i(t)$ and Key vector $K^i(s)$. One unique characteristic of heterogeneous graphs is that there may exist different edge types (relations) between a node type pair, e.g., $\tau(s)$ and $\tau(t)$. Therefore, unlike the vanilla Transformer that directly calculates the dot product between the Query and Key vectors, we keep a distinct edge-based matrix $W^{ATT}_{\phi(e)} \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$ for each edge type $\phi(e)$. In doing so, the model can capture different semantic relations even between

the same node type pairs. Moreover, since not all the relationships contribute equally to the target nodes, we add a prior tensor $\mu \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{R}|\times|\mathcal{A}|}$ to denote the general significance of each meta relation triplet, serving as an adaptive scaling to the attention.

Finally, we concatenate h attention heads together to get the attention vector for each node pair. Then, for each target node t, we gather all attention vectors from its neighbors $N(t)$ and conduct softmax, making it fulfill $\sum_{\forall s\in N(t)} \textbf{Attention}_{HGT}(s,e,t) = \mathbf{1}_{h\times 1}$.

3.3 Heterogeneous Message Passing

Parallel to the calculation of mutual attention, we pass information from source nodes to target nodes (See Figure 2 (2)). Similar to the attention process, we would like to incorporate the meta relations of edges into the message passing process to alleviate the distribution differences of nodes and edges of different types. For a pair of nodes $e = (s,t)$, we calculate its multi-head **Message** by:

$$\textbf{Message}_{HGT}(s,e,t) = \Big\Vert_{i\in[1,h]} \mathit{MSG\text{-}head}^i(s,e,t) \tag{4}$$

$$\mathit{MSG\text{-}head}^i(s,e,t) = \text{M-Linear}^i_{\tau(s)}\Big(H^{(l-1)}[s]\Big)\, W^{MSG}_{\phi(e)}$$

To get the i-th message head $\mathit{MSG\text{-}head}^i(s,e,t)$, we first project the $\tau(s)$-type source node s into the i-th message vector with a linear projection $\text{M-Linear}^i_{\tau(s)} : \mathbb{R}^d \to \mathbb{R}^{\frac{d}{h}}$. It is then followed by a matrix $W^{MSG}_{\phi(e)} \in \mathbb{R}^{\frac{d}{h}\times\frac{d}{h}}$ for incorporating the edge dependency. The final step is to concat all h message heads to get the $\textbf{Message}_{HGT}(s,e,t)$ for each node pair.

3.4 Target-Specific Aggregation

With the heterogeneous multi-head attention and message calculated, we need to aggregate them from the source nodes to the target node (See Figure 2 (3)). Note that the softmax procedure in Eq. 3 has made the sum of each target node t's attention vectors to one, we can thus simply use the attention vector as the weight to average the corresponding messages from the source nodes and get the updated vector $\widetilde{H}^{(l)}[t]$ as:

$$\widetilde{H}^{(l)}[t] = \bigoplus_{\forall s\in N(t)} \Big(\textbf{Attention}_{HGT}(s,e,t)\cdot \textbf{Message}_{HGT}(s,e,t)\Big).$$

This aggregates information to the target node t from all its neighbors (source nodes) of different feature distributions.

The final step is to map target node t's vector back to its type-specific distribution, indexed by its node type $\tau(t)$. To do so, we apply a linear projection $\text{A-Linear}_{\tau(t)}$ to the updated vector $\widetilde{H}^{(l)}[t]$, followed by residual connection [8] as:

$$H^{(l)}[t] = \text{A-Linear}_{\tau(t)}\Big(\sigma\big(\widetilde{H}^{(l)}[t]\big)\Big) + H^{(l-1)}[t]. \tag{5}$$

In this way, we get the l-th HGT layer's output $H^{(l)}[t]$ for the target node t. Due to the "small-world" property of real-world graphs, stacking the HGT blocks for L layers (L being a small value) can enable each node reaching a large proportion of nodes—with different types and relations—in the full graph. That is, HGT generates a highly contextualized representation $H^{(L)}$ for each node, which can be fed into any models to conduct downstream heterogeneous network tasks, such as node classification and link prediction.

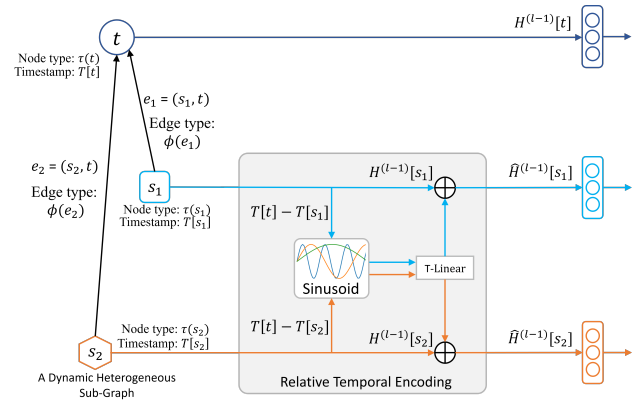


Figure 3: Relative Temporal Encoding (RTE) to model graph dynamic. Nodes are associated with timestamps $T(\cdot)$. After the RTE process, the temporal augmented representations are fed to the HGT model.

Through the whole model architecture, we highly rely on using the **meta relation**—$\langle\tau(s),\phi(e),\tau(t)\rangle$—to parameterize the weight matrices separately. This can be interpreted as a trade-off between the model capacity and efficiency. Compared with the vanilla Transformer, our model distinguishes the operators for different relations and thus is more capable to handle the distribution differences in heterogeneous graphs. Compared with existing models that keep a distinct matrix for each meta relation as a whole, HGT's triplet parameterization can better leverage the heterogeneous graph schema to achieve parameter sharing. On one hand, relations with few occurrences can benefit from such parameter sharing for fast adaptation and generalization. On the other hand, different relationships' operators can still maintain their specific characteristics by using a much smaller parameter set.

3.5 Relative Temporal Encoding

By far, we present HGT—a graph neural network for modeling heterogeneous graphs. Next, we introduce the Relative Temporal Encoding (RTE) technique for HGT to handle graph dynamic.

The traditional way to incorporate temporal information is to construct a separate graph for each time slot. However, such a procedure may lose a large portion of structural dependencies across different time slots. Meanwhile, the representation of a node at time t might rely on edges that happen at other time slots. Therefore, a proper way to model dynamic graphs is to maintain all the edges happening at different times and allow nodes and edges with different timestamps to interact with each other.

In light of this, we propose the Relative Temporal Encoding (RTE) mechanism to model the dynamic dependencies in heterogeneous graphs. RTE is inspired by Transformer's positional encoding method [15, 21], which has been shown successful to capture the sequential dependencies of words in long texts.

Specifically, given a source node s and a target node t, along with their corresponding timestamps $T(s)$ and $T(t)$, we denote the relative time gap $\Delta T(t,s) = T(t) - T(s)$ as an index to get a relative temporal encoding $RTE(\Delta T(t,s))$. Noted that the training dataset

will not cover all possible time gaps, and thus RTE should be capable of generalizing to unseen times and time gaps. Therefore, we adopt a fixed set of sinusoid functions as basis, with a tunable linear projection T-Linear*: $\mathbb{R}^d \to \mathbb{R}^d$ as RTE:

$$Base\big(\Delta T(t,s), 2i\big) = sin\Big(\Delta T_{t,s}/10000^{\frac{2i}{d}}\Big) \tag{6}$$

$$Base\big(\Delta T(t,s), 2i+1\big) = cos\Big(\Delta T_{t,s}/10000^{\frac{2i+1}{d}}\Big) \tag{7}$$

$$RTE\big(\Delta T(t,s)\big) = \text{T-Linear}\Big(Base(\Delta T_{t,s})\Big) \tag{8}$$

Finally, the temporal encoding relative to the target node t is added to the source node s' representation as follows:

$$\widehat{H}^{(l-1)}[s] = H^{(l-1)}[s] + RTE\big(\Delta T(t,s)\big) \tag{9}$$

In this way, the temporal augmented representation $\widehat{H}^{(l-1)}$ will capture the relative temporal information of source node s and target node t. The RTE procedure is illustrated in the Figure 3.

4 WEB-SCALE HGT TRAINING

In this section, we present HGT's strategies for training Web-scale heterogeneous graphs with dynamic information, including an efficient Heterogeneous Mini-Batch Graph Sampling algorithm—HGSampling—and an inductive timestamp assignment method.

4.1 HGSampling

The full-batch GNN [9] training requires the calculation of all node representations per layer, making it not scalable for Web-scale graphs. To address this issue, different sampling-based methods [1, 2, 7, 29] have been proposed to train GNNs on a subset of nodes. However, directly using them for heterogeneous graphs is prone to get sub-graphs that are extremely imbalanced regarding different node types, due to that the degree distribution and the total number of nodes for each type can vary dramatically.

To address this issue, we propose an efficient Heterogeneous Mini-Batch Graph Sampling algorithm—HGSampling—to enable both HGT and traditional GNNs to handle Web-scale heterogeneous graphs. HGSampling is able to 1) keep a similar number of nodes and edges for each type and 2) keep the sampled sub-graph dense to minimize the information loss and reduce the sample variance.

Algorithm 1 outlines the HGSampling algorithm. Its basic idea is to keep a separate node budget $B[\tau]$ for each node type τ and to sample an equal number of nodes per type with an importance sampling strategy to reduce variance. Given node t already sampled, we add all its direct neighbors into the corresponding budget with Algorithm 2, and add t's normalized degree to these neighbors in line 8, which will then be used to calculate the sampling probability. Such normalization is equivalent to accumulate the random walk probability of each sampled node to its neighborhood, avoiding the sampling being dominated by high-degree nodes. Intuitively, the higher such value is, the more a candidate node is correlated with the currently sampled nodes, and thus should be given a higher probability to be sampled.

*For simplicity, we denote a linear projection $\text{L} : \mathbb{R}^a \to \mathbb{R}^b$ as a function to conduct linear transformation to vector $x \in \mathbb{R}^a$ as: $\text{L}(x) = Wx + b$, where matrix $W \in \mathbb{R}^{a+b}$ and bias $b \in \mathbb{R}^b$. W and b are learnable parameters for L.

Algorithm 1 Heterogeneous Mini-Batch Graph Sampling

Require: Adjacency matrix A for each $\langle \tau(s), \phi(e), \tau(t) \rangle$ relation pair; Output node Set OS; Sample number n per node type; Sample depth L.

Ensure: Sampled node set NS; Sampled adjacency matrix $\hat{A}$.

1: $NS \leftarrow OS$ // Initialize sampled node set as output node set.
2: Initialize an empty Budget B storing nodes for each node type with normalized degree.
3: **for** $t \in NS$ **do**
4: Add-In-Budget(B, t, A, NS) // Add neighbors of t to B.
5: **end for**
6: **for** $l \leftarrow 1$ to L **do**
7: **for** source node type $\tau \in B$ **do**
8: **for** source node $s \in B[\tau]$ **do**
9: $prob^{(l-1)}[\tau][s] \leftarrow \frac{B[\tau][s]^2}{\|B[\tau]\|_2^2}$ // Calculate sampling probability for each source node s of node type τ.
10: **end for**
11: Sample n nodes $\{t_i\}_{i=1}^n$ from $B[\tau]$ using $prob^{(l-1)}[\tau]$.
12: **for** $t \in \{t_i\}_{i=1}^n$ **do**
13: $OS[\tau].add(t)$ // Add node t into Output node set.
14: Add-In-Budget(B, t, A, NS) // Add neighbors of t to B.
15: $B[\tau].pop(t)$ // Remove sampled node t from Budget.
16: **end for**
17: **end for**
18: **end for**
19: Reconstruct the sampled adjacency matrix $\hat{A}$ among the sampled nodes OS from A.
20: **return** OS and $\hat{A}$;

After the budget is updated, we then calculate the sampling probability in Algorithm 1 line 9, where we calculate the square of the cumulative normalized degree of each node s in each budget. As proved in [29], using such sampling probability can reduce the sampling variance. Then, we sample n nodes in type τ by using the calculated probability, add them into the output node set, update its neighborhood to the budget, and remove it out of the budget in lines 12–15. Repeating such procedure for L times, we get a sampled sub-graph with L depth from the initial nodes. Finally, we reconstruct the adjacency matrix among the sampled nodes. By using the above algorithm, the sampled sub-graph contains a similar number of nodes per type (based on the separate node budget), and is sufficiently dense to reduce the sampling variance (based on the normalized degree and importance sampling), making it suitable for training GNNs on Web-scale heterogeneous graphs.

4.2 Inductive Timestamp Assignment

Till now we have assumed that each node t is assigned with a timestamp $T(t)$. However, in real-world heterogeneous graphs, many nodes are not associated with a fixed time. Therefore, we need to assign different timestamps to it. We denote these nodes as plain nodes. For example, the WWW conference is held in both 1974 and 2019, and the WWW node in these two years has dramatically different research topics. Consequently, we need to decide which timestamp(s) to attach to the WWW node.

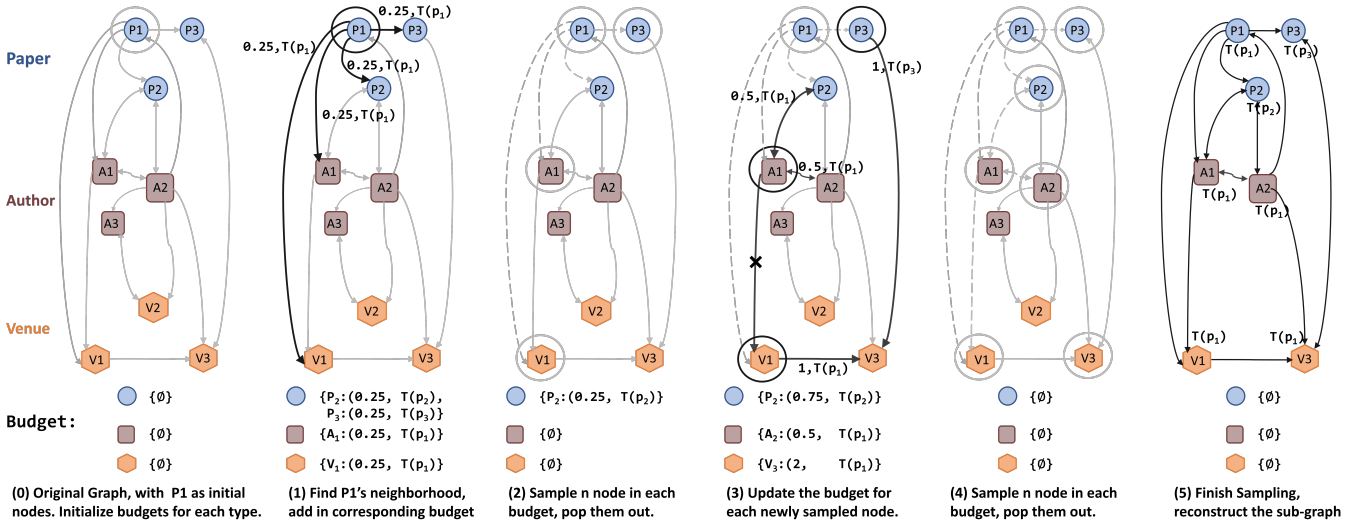


Figure 4: HGSampling with Inductive Timestamp Assignment.

Algorithm 2 Add-In-Budget

Require: Budget B storing nodes for each type with normalized degree; Added node t; Adjacency matrix A for each $\langle \tau(s), \phi(e), \tau(t) \rangle$ relation pair; Sampled node set NS.
Ensure: Updated Budget B.

1: **for** each possible source node type τ and edge type ϕ **do**
2: $\hat{D}_t \leftarrow 1 \;/\; len\left(A_{\langle \tau, \phi, \tau(t) \rangle}[t]\right)$ // get normalized degree of added node t regarding to $\langle \tau, \phi, \tau(t) \rangle$.
3: **for** source node s in $A_{\langle \tau, \phi, \tau(t) \rangle}[t]$ **do**
4: **if** s has not been sampled ($s \notin NS$) **then**
5: **if** s has no timestamp **then**
6: $s.time = t.time$ // Inductively inherit timestamp.
7: **end if**
8: $B[\tau][s] \leftarrow B[\tau][s] + \hat{D}_t$ // Add candidate node s to budget B with target node t's normalized degree.
9: **end if**
10: **end for**
11: **end for**
12: **return** Updated Budget B

There also exist event nodes in heterogeneous graphs that have an explicit timestamp associated with them. For example, the paper node should be associated with its publication behavior and therefore attached to its publication date.

We propose an inductive timestamp assignment algorithm to assign plain nodes timestamps based on event nodes that they are linked with. The algorithm is shown in Algorithm 2 line 6. The idea is that plan nodes inherit the timestamps from event nodes. We examine whether the candidate source node is an event node. If yes, like a paper published at a specific year, we keep its timestamp for capturing temporal dependency. If no, like a conference that can be associated with any timestamp, we inductively assign the associated node's timestamp, such as the published year of its paper, to this plain node. In this way, we can adaptively assign timestamps during the sub-graph sampling procedure.

5 EVALUATION

In this section, we evaluate the proposed Heterogeneous Graph Transformer on three heterogeneous academic graph datasets. We conduct the Paper-Field prediction, Paper-Venue prediction, and Author Disambiguation tasks. We also take case studies to demonstrate how HGT can automatically learn and extract meta paths that are important for downstream tasks[†].

5.1 Web-Scale Datasets

To examine the performance of the proposed model and its real-world applications, we use the Open Academic Graph (OAG) [16, 20, 28] as our experimental basis. OAG consists of more than 178 million nodes and 2.236 billion edges—the largest publicly available heterogeneous academic dataset. In addition, all papers in OAG are associated with their publication dates, spanning from 1900 to 2019.

To test the generalization of the proposed model, we also construct two domain-specific subgraphs from OAG: the Computer Science (CS) and Medicine (Med) academic graphs. The graph statistics are listed in Table 1, in which P–A, P–F, P–V, A–I, and P–P denote the edges between paper and author, paper and field, paper and venue, author and institute, and the citation links between two papers.

Both the CS and Med graphs contain tens of millions of nodes and hundreds of millions of edges, making them at least one magnitude larger than the other CS (e.g., DBLP) and Med (e.g., Pubmed) academic datasets that are commonly used in existing heterogeneous GNN and heterogeneous graph mining studies. Moreover, the three datasets used are far more distinguishable than previously wide-adopted small citation graphs used in GNN studies, such as

[†]The dataset and code are publicly available at https://github.com/acbull/pyHGT.

Dataset	#nodes	#edges	#papers	#authors	#fields	#venues	#institutes	#P-A	#P-F	#P-V	#A-I	#P-P
CS	11,732,027	107,263,811	5,597,605	5,985,759	119,537	27,433	16,931	15,571,614	47,462,559	5,597,606	7,190,480	31,441,552
Med	51,044,324	451,468,375	21,931,587	28,779,507	289,930	25,044	18,256	85,620,479	149,728,483	21,931,588	28,779,507	165,408,318
OAG	178,663,927	2,236,196,802	89,606,257	88,364,081	615,228	53,073	25,288	300,853,688	657,049,405	89,606,258	167,449,933	1,021,237,518

Table 1: Open Academic Graph (OAG) Statistics.

Cora, Citeseer, and Pubmed [9, 22], which only contain thousands of nodes.

There are totally five node types: 'Paper', 'Author', 'Field', 'Venue', and 'Institute'. The 'Field' nodes in OAG are categorized into six levels from L_0 to L_5, which are organized with a hierarchical tree. Therefore, we differentiate the 'Paper–Field' edges corresponding to the field level.

In addition, we differentiate the different author orders (i.e., the first author, the last one, and others) and venue types (i.e., journal, conference, and preprint) as well. Finally, the 'Self' type corresponds to the self-loop connection, which is widely added in GNN architectures. Except the 'Self' relationship, which are symmetric, all other relation types ϕ have a reverse relation type ϕ^{-1}.

5.2 Experimental Setup

Tasks and Evaluation. We evaluate the HGT model on four different real-world downstream tasks: the prediction of Paper–Field (L_1), Paper–Field (L_2), and Paper–Venue, and Author Disambiguation. The goal of the first three node classification tasks is to predict the correct L_1 and L_2 fields that each paper belongs to or the venue it is published at, respectively. We use different GNNs to get the contextual node representation of the paper and use a softmax output layer to get its classification label. For author disambiguation, we select all the authors with the same name and their associated papers. The task is to conduct link prediction between these papers and candidate authors. After getting the paper and author node representations from GNNs, we use a Neural Tensor Network to get the probability of each author-paper pair to be linked.

For all tasks, we use papers published before the year 2015 as the training set, papers between 2015 and 2016 for validation, and papers between 2016 and 2019 as testing. We choose NDCG and MRR, which are two widely adopted ranking metrics [10, 11], as the evaluation metrics. All models are trained for 5 times and, the mean and standard variance of test performance are reported.

Baselines. We compare HGT with two classes of state-of-art graph neural networks. All baselines as well as our own model, are implemented via the PyTorch Geometric (PyG) package [5].

The first class of GNN baselines is designed for homogeneous graphs, including:

- Graph Convolutional Networks (GCN) [9], which simply averages the neighbor's embedding followed by linear projection. We use the implementation provided in PyG.
- Graph Attention Networks (GAT) [22], which adopts multi-head additive attention on neighbors. We use the implementation provided in PyG.

The second class considered is several dedicated heterogeneous GNNs as baselines, including:

- Relational Graph Convolutional Networks (RGCN) [14], which keeps a different weight for each relationship, i.e., a relation triplet. We use the implementation provided in PyG.
- Heterogeneous Graph Neural Networks (HetGNN) [27], which adopts different Bi-LSTMs for different node type for aggregating neighbor information. We re-implement this model in PyG following the authors' official code.
- Heterogeneous Graph Attention Networks (HAN) [23] design hierarchical attentions to aggregate neighbor information via different meta paths. We re-implement this model in PyG following the authors' official code.

In addition, to systematically analyze the effectiveness of the two major components of HGT, i.e., Heterogeneous weight parameterization (Heter) and Relative Temporal Encoding (RTE), we conduct an ablation study, but comparing with models that remove these components. Specifically, we use $-Heter$ to denote models that uses the same set of weights for all meta relations, and use $-RTE$ to denote models that doesn't include relative temporal encoding. By considering all the permutations, we have: $\text{HGT}^{-RTE}_{-Heter}$, $\text{HGT}^{+RTE}_{-Heter}$, $\text{HGT}^{-RTE}_{+Heter}$ and $\text{HGT}^{+RTE}_{+Heter}$ [‡].

We use our HGSampling algorithm proposed in Section 4 for all baseline GNNs to handle the large-scale OAG graph. To avoid data leakage, we remove out the links we aim to predict (e.g., the Paper-Field link as the label) from the sub-graph.

Input Features. As we don't assume the feature of each node type belongs to the same distribution, we are free to use the most appropriate features to represent each type of nodes. For each paper, we use a pre-trained XLNet [24, 25] to get the representation of each word in its title. We then average them weighted by each word's attention to get the title representation for each paper. The initial feature of each author is then simply an average of his/her published papers' representations. For the field, venue, and institute nodes, we use the metapath2vec model [3] to train their node embeddings by reflecting the heterogeneous network structures.

The homogeneous GNN baselines assume the node features belong to the same distribution, while our feature extraction doesn't fulfill this assumption. To make a fair comparison, we add an adaptation layer between the input features and all used GNNs. This module simply conducts different linear projections for nodes of different types. Such a procedure can be regarded to map heterogeneous data into the same distribution, which is also adopted in literature [23, 27].

Implementation Details. We use 256 as the hidden dimension throughout the neural networks for all baselines. For all multi-head attention-based methods, we set the head number as 8. All GNNs keep 3 layers so that the receptive fields of each network are exactly

[‡]Unless other stated, HGT refers to $\text{HGT}^{+RTE}_{+Heter}$.

GNN Models			GCN [9]	RGCN [14]	GAT [22]	HetGNN [27]	HAN [23]	HGT$^{-RTE}_{-Heter}$	HGT$^{+RTE}_{-Heter}$	HGT$^{-RTE}_{+Heter}$	HGT$^{+RTE}_{+Heter}$
# of Parameters			1.69M	8.80M	1.69M	8.41M	9.45M	3.12M	3.88M	7.44M	8.20M
Batch Time			0.46s	1.24s	0.97s	1.35s	2.27s	1.11s	1.14s	1.48s	1.50s
CS	Paper–Field (L_1)	NDCG	.608±.062	.603±.065	.622±.071	.612±.063	.618±.058	.662±.051	.689±.042	.705±.036	**.718±.014**
		MRR	.679±.069	.683±.056	.694±.065	.689±.060	.691±.051	.751±.036	.779±.027	.799±.023	**.823±.019**
	Paper–Field (L_2)	NDCG	.344±.021	.322±.053	.357±.058	.346±.071	.352±.051	.362±.048	.371±.043	.379±.047	**.403±.041**
		MRR	.353±.053	.340±.061	.382±.057	.373±.051	.388±.065	.394±.072	.397±.064	.414±.076	**.439±.078**
	Paper–Venue	NDCG	.406±.081	.412±.076	.437±.082	.431±.074	.449±.072	.456±.069	.461±.066	.468±.074	**.473±.054**
		MRR	.215±.066	.216±.105	.239±.089	.245±.069	.254±.074	.258±.085	.265±.090	.275±.089	**.288±.088**
	Author Disambiguation	NDCG	.826±.039	.835±.042	.864±.051	.850±.056	.859±.053	.867±.048	.875±.046	.886±.048	**.894±.034**
		MRR	.661±.045	.665±.054	.694±.052	.668±.061	.688±.049	.703±.036	.712±.032	.727±.038	**.732±.038**
Med	Paper–Field (L_1)	NDCG	.560±.056	.571±.061	.584±.076	.598±.068	.607±.054	.654±.048	.667±.045	.683±.037	**.709±.029**
		MRR	.465±.055	.470±.082	.493±.069	.509±.054	.575±.057	.620±.066	.642±.062	.659±.055	**.688±.048**
	Paper–Field (L_2)	NDCG	.334±.035	.337±.051	.344±.063	.342±.048	.350±.059	.359±.053	.365±.047	.374±.050	**.384±.046**
		MRR	.337±.061	.343±.063	.370±.058	.373±.061	.379±.052	.385±.071	.397±.069	.408±.071	**.417±.074**
	Paper–Venue	NDCG	.377±.059	.383±.062	.388±.065	.412±.057	.416±.068	.421±.083	.432±.078	**.446±.083**	.445±.085
		MRR	.211±.045	.217±.058	.244±.091	.259±.072	.271±.056	.277±.081	.282±.085	.288±.074	**.291±.062**
	Author Disambiguation	MRR	.776±.042	.779±.048	.828±.044	.824±.058	.834±.056	.838±.047	.844±.041	.864±.043	**.871±.040**
		NDCG	.614±.051	.625±.049	.663±.046	.659±.061	.667±.053	.683±.055	.691±.046	.708±.041	**.718±.043**
OAG	Paper–Field (L_1)	NDCG	.508±.141	.511±.128	.534±.103	.543±.084	.544±.096	.571±.089	.578±.086	.595±.089	**.615±.084**
		MRR	.556±.136	.565±.105	.610±.096	.616±.076	.622±.092	.649±.081	.657±.078	.675±.082	**.702±.081**
	Paper–Field (L_2)	NDCG	.318±.074	.328±.046	.339±.049	.336±.062	.342±.051	.350±.045	.354±.046	.358±.052	**.367±.048**
		MRR	.322±.067	.332±.052	.348±.045	.350±.053	.358±.049	.362±.057	.369±.058	.371±.064	**.378±.071**
	Paper–Venue	NDCG	.302±.066	.313±.051	.317±.057	.309±.071	.327±.062	.334±.058	.341±.059	.353±.064	**.355±.062**
		MRR	.194±.070	.193±.047	.196±.052	.192±.059	.214±.067	.229±.061	.233±.060	.243±.048	**.247±.061**
	Author Disambiguation	NDCG	.738±.042	.755±.048	.797±.044	.803±.058	.821±.056	.835±.043	.841±.041	.847±.043	**.852±.048**
		MRR	.612±.064	.619±.057	.645±.063	.649±.052	.660±.049	.668±.059	.674±.058	.683±.066	**.688±.054**

Table 2: Experimental results of different methods over the three datasets.

the same. All baselines are optimized via the AdamW optimizer [13] with the Cosine Annealing Learning Rate Scheduler [12]. For each model, we train it for 200 epochs and select the one with the lowest validation loss as the reported model. We use the default parameters used in GNN literature and donot tune hyper-parameters.

5.3 Experimental Results

We summarize the experimental results of the proposed model and baselines on three datasets in Table 2. All experiments for the four tasks are evaluated in terms of NDCG and MRR.

The results show that in terms of both metrics, the proposed HGT significantly and consistently outperforms all baselines for all tasks on all datasets. Take, for example, the Paper–Field (L_1) classification task on OAG, HGT achieves relative performance gains over baselines by 15–19% in terms of NDCG and 18–21% in terms of MRR (i.e., the performance gap divided by the baseline performance). When compared to HAN—the best baseline for most of the cases, the average relative NDCG improvements of HGT on the CS, Med and OAG datasets are 11%, 10% and 8%, respectively.

Overall, we observe that on average, HGT outperforms GCN, GAT, RGCN, HetGNN, and HAN by 20% for the four tasks on all three large-scale datasets. Moreover, HGT has fewer parameters and comparable batch time than all the heterogeneous graph neural network baselines, including RGCN, HetGNN, and HAN. This suggests that by modeling heterogeneous edges according to their meta relation schema, we are able to have better generalization with fewer resource consumption.

Ablation Study. The core component in HGT are the heterogeneous weight parameterization (Heter) and Relative Temporal Encoding (RTE). To further analyze their effects, we conduct an ablation study by removing them from HGT. Specifically, the model that removes heterogeneous weight parameterization, i.e., $\text{HGT}^{+RTE}_{-Heter}$, drops 4% of performance compared with the full model $\text{HGT}^{+RTE}_{+Heter}$. By removing RTE (i.e., $\text{HGT}^{-RTE}_{+Heter}$), the performance has a 2% drop. The ablation study shows the significance of parameterizing with meta relations and using Relative Temporal Encoding.

In addition, we also try to implement a baseline that keeps a unique weight matrix for each relation. However, such a baseline contains too many parameters so that our experimental setting doesn't have enough GPU memory to optimize it. This also indicates that using the meta relation to parameterize weight matrices can achieve competitive performance with fewer resources.

5.4 Case Study

To further evaluate how Relative Temporal Encoding (RTE) can help HGT to capture graph dynamics, we conduct a case study showing the evolution of conference topic. We select 100 conferences in computer science with the highest citations, assign them three different timestamps, i.e., 2000, 2010 and 2020, and construct sub-graphs initialized by them. Using a trained HGT, we can get the representations for these conferences, with which we can calculate the euclidean distances between them. We select WWW, KDD, and NeurIPS as illustration. For each of them, we pick the top-5 most similar conferences (i.e., the one with the smallest euclidean distance) to show how the conference's topics evolve over time.

As shown in Table 3, these venues' relationships have changed from 2000 to 2020. For example, WWW in 2000 was more related to some database conferences, i.e., SIGMOD and VLDB, and some networking conferences, i.e., NSDI and GLOBECOM. However, WWW in 2020 would become more related to some data mining and information retrieval conferences (KDD, SIGIR, and WSDM), in addition to SIGMOD and GLOBECOM. Also, KDD in 2000 was more related to traditional database and data mining venues, while in 2020 it will tend to correlate with a variety of topics, i.e. machine learning (NeurIPS), database (SIGMOD), Web (WWW), AI (AAAI), and NLP (EMNLP). Additionally, our HGT model can capture the difference brought by new conferences. For example, NeurIPS in 2020 would relate with ICLR, which is a newly organized deep learning conference. This case study shows that the relative temporal encoding can help capture the temporal evolution of the heterogeneous academic graphs.

5.5 Visualize Meta Relation Attention

To illustrate how the incorporated meta relation schema can benefit the heterogeneous message passing process, we pick the schema that has the largest attention value in each of the first two HGT layers and plot the meta relation attention hierarchy tree in Figure 5. For example, to calculate a paper's representation, ⟨Paper, *is_published_at*, Venue, $is\_published\_at^{-1}$, Paper⟩, ⟨Paper, $has\_L_2\_field\_of$, Field, $has\_L_5\_field\_of^{-1}$, Paper⟩, and ⟨Institute, $is\_affiliated\_with^{-1}$, Author, $is\_first\_author\_of$, Paper⟩ are the three most important meta relation sequences, which can be regarded as meta paths *PVP, PFP,* and *IAP*, respectively. Note that these meta paths and their importance are automatically learned from the data without manual design. Another example of calculating an author node's representation is shown on the right. Such visualization demonstrates that Heterogeneous Graph Transformer is capable of implicitly learning to construct important meta paths for specific downstream tasks, without manual customization.

Venue	Time	Top−5 Most Similar Venues
WWW	2000	SIGMOD, VLDB, NSDI, GLOBECOM, SIGIR
	2010	GLOBECOM, KDD, CIKM, SIGIR, SIGMOD
	2020	KDD, GLOBECOM, SIGIR, WSDM, SIGMOD
KDD	2000	SIGMOD, ICDE, ICDM, CIKM, VLDB
	2010	ICDE, WWW, NeurIPS, SIGMOD, ICML
	2020	NeurIPS, SIGMOD, WWW, AAAI, EMNLP
NeurIPS	2000	ICCV, ICML, ECCV, AAAI, CVPR
	2010	ICML, CVPR, ACL, KDD, AAAI
	2020	ICML, CVPR, ICLR, ICCV, ACL

Table 3: Temporal Evolution of Conference Similarity.

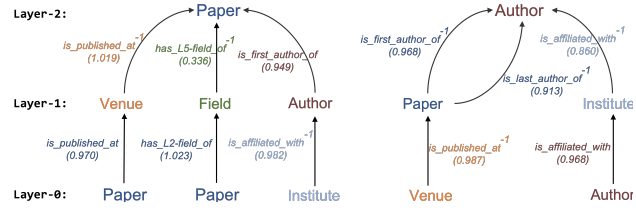


Figure 5: Hierarchy of the learned meta relation attention.

6 CONCLUSION

In this paper, we propose the Heterogeneous Graph Transformer (HGT) architecture for modeling Web-scale heterogeneous and dynamic graphs. To model heterogeneity, we use the meta relation $\langle \tau(s), \phi(e), \tau(t) \rangle$ to decompose the interaction and transform matrices, enabling the model to have the similar modeling capacity with fewer resources. To capture graph dynamics, we present the relative temporal encoding (RTE) technique to incorporate temporal information using limited computational resources. To conduct efficient and scalable training of HGT on Web-scale data, we design the heterogeneous Mini-Batch graph sampling algorithm—HGSampling. We conduct comprehensive experiments on the Open Academic Graph, and show that the proposed HGT model can capture both heterogeneity and outperforms all the state-of-the-art GNN baselines on various downstream tasks.

In the future, we will explore whether HGT is able to generate heterogeneous graphs, e.g., predict new papers and their titles, and whether we can pre-train HGT to benefit tasks with scarce labels.

Acknowledgements. We would like to thank Xiaodong Liu for helpful discussions. This work is partially supported by NSF III-1705169, NSF CAREER Award 1741634, NSF#1937599, Okawa Foundation Grant, and Amazon Research Award.

REFERENCES

[1] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR'18*.

[2] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*. 941–949.

[3] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD '17*.

[4] Yuxiao Dong, Hao Ma, Zhihong Shen, and Kuansan Wang. 2017. A Century of Science: Globalization of Scientific Collaborations, Citations, and Innovations. In *KDD '17*. ACM, 1437–1446.

[5] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *ICLR 2019 Workshop: Representation Learning on Graphs and Manifolds* (2019).

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.

[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS'17*.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR'16*.

[9] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR'17*.

[10] Hang Li. 2014. *Learning to Rank for Information Retrieval and Natural Language Processing, Second Edition*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00607ED2V01Y201410HLT026

[11] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer.

[12] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR'17*.

[13] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR'19*.

[14] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC'2018*.

[15] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *NAACL-HLT*. 464–468.

[16] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW Companion 2015*.

[17] Yizhou Sun and Jiawei Han. 2012. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Morgan & Claypool Publishers.

[18] 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. In *VLDB '11*.

[19] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S. Yu, and Xiao Yu. 2012. Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. In *KDD'12*.

[20] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS'17*.

[22] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR'18*.

[23] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous Graph Attention Network. In *KDD'19*. 2022–2032.

[24] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, RÃĬmi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Transformers: State-of-the-art Natural Language Processing. arXiv:cs.CL/1910.03771

[25] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS'19*.

[26] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph Transformer Networks. In *NeurIPS'19*.

[27] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *WWW'19*.

[28] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, and Kuansan Wang. 2019. OAG: Toward Linking Large-scale Heterogeneous Entity Graphs. In *KDD'19*.

[29] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *NeurIPS'19*.