Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

TAHAN: Type-Aware Heterogeneous Graph Attention Network

Amine Lakhder

amine.lakhder@usmba.ac.ma

Sidi Mohamed Ben Abdellah University

Fahd Kalloubi

Cadi Ayyad University

Lamiae Azizi

Université Mohammed VI Polytechnique

Imad Kissami

Université Mohammed VI Polytechnique

Abdelhak Boulaalam

Sidi Mohamed Ben Abdellah University

Research Article

Keywords: Heterogeneous Graph Neural Networks, Deep Learning on Graphs, Relational Learning, Type-Aware Aggregation

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11272420/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

TAHAN: Type-Aware Heterogeneous Graph Attention Network

Amine Lakhder[1*], Fahd Kalloubi[2,3], Lamiae Azizi[2], Imad Kissami[2], Abdelhak Boulaalam[1]

[1*]National School of Applied Sciences, Sidi Mohamed Ben Abdellah University, Fez, 30000, Fès-Meknès, Morocco.

[2]College of Computing, Mohammed VI Polytechnic University, Benguerir, 43150, Marrakesh-Safi, Morocco.

[3]Computer Science and Smart Systems Laboratory, Faculty of Sciences, Cadi Ayyad University, Marrakech, 40000, Morocco.

*Corresponding author(s). E-mail(s): amine.lakhder@usmba.ac.ma;
Contributing authors: kalloubi.f@ucd.ac.ma; lamiae.azizi@um6p.ma;
imad.kissami@um6p.ma; abdelhak.boulaalam@usmba.ac.ma;

Abstract

Learning from heterogeneous graphs remains challenging when structural complexity exceeds the modeling capacity of existing heterogeneous graph neural networks (HGNNs). Although current HGNNs capture structural and semantic relations, many treat node types as static categorical labels and underuse their contextual influence during message passing. Metapath-based aggregation can also overlook the semantic contributions of intermediate node types. These limitations can reduce the use of type-dependent contextual signals, particularly for sparsely or weakly connected entity types. We propose Type-Aware Heterogeneous Graph Attention Network (**TAHAN**), which incorporates type-sensitive and context-dependent information into heterogeneous neighborhood aggregation. TAHAN first collects neighbors along metapath prefixes and across multiple neighborhood levels, merges them across metapaths, and organizes them into type-specific neighborhoods. It then applies *type-specific attention* to capture contributions from neighbors within each type, followed by *type-aware attention* to adaptively fuse the resulting representations. This two-stage aggregation produces expressive, type-sensitive node embeddings. Experiments on node classification, clustering, and link prediction demonstrate competitive performance compared with state-of-the-art HGNNs. Component-level analyses further illustrate the contribution and interpretability of type-aware modeling, offering insights for future research.

Keywords: Heterogeneous Graph Neural Networks, Deep Learning on Graphs, Relational Learning, Type-Aware Aggregation

1 Introduction

Graph data has become increasingly prevalent over the past decades, capturing the intricate relationships and structures inherent in many real-world systems. From consumer-facing applications such as social networks [35] [31] [4], recommendation systems [51] [32], databases [22], and navigation tools [21] [57] to complex domains such as healthcare and biology [53] [38] [7] [19] [48] [42], graphs provide a powerful framework for modeling interconnected data. Graphs represent not only individual entities but also their interactions, making them indispensable for a wide array of tasks such as predicting user preferences [45], detecting fraudulent behavior [43], and diagnosing medical conditions [12].

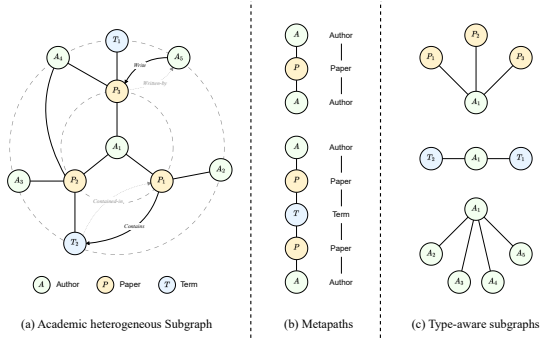


Fig. 1 An illustrative example of a heterogeneous graph (DBLP) and the neighborhood construction process. (a) The local structure of node A1 in the heterogeneous DBLP graph, showing its connections with different entity types. (b) Predefined metapaths that guide the selection of relevant relationships. (c) Semantic subgraphs constructed based on the neighborhood construction process, capturing meaningful structural and relational patterns

A particularly important subclass of graphs is heterogeneous graphs (HGs), where nodes and edges can belong to multiple types and possess diverse attributes and semantics [14]. This diversity makes heterogeneous graphs especially useful for representing rich and multifaceted datasets, such as knowledge graphs, social networks, academic citation networks and biological networks [48] [47] [16] [19] [10], in which entities (nodes) and their relationships (edges) reflect diverse roles, structures, and contextual meanings. This complexity enhances the representational power of these heterogeneous graphs, but it also introduces unique challenges for analysis, modeling, and learning. Thus, traditional graph-based approaches fall short in handling this heterogeneity, resulting in suboptimal models that fail to fully exploit and capture the complex interactions present in these networks.

In recent years, graph neural networks (GNNs) have emerged as the de facto standard for advancing graph mining and representation learning over graph-like data [29] [17] [39] [13] [2]. These models have demonstrated significant success across a range of tasks, from node classification and link prediction to graph-level understanding [44] [58] [59]. GNNs leverage neighborhood information and learn how nodes interact with one another through message-passing mechanisms [10]. However, most traditional GNN architectures are designed with homogeneous graphs in mind, which assume uniformity in node and edge types. This assumption works well when all nodes and edges in the graph are of the same type, but it becomes problematic in heterogeneous graphs, where the structure and semantics of nodes and edges differ substantially. Adapting GNNs to heterogeneous graphs requires addressing challenges such as managing the diversity of node and edge types, capturing type-specific semantics, and scaling to the often higher complexity of these data structures. Additionally, ensuring that different types of nodes and edges contribute appropriately to the representation learning process is critical for achieving high-quality embeddings.

To address these challenges, Heterogeneous Graph Neural Networks (HGNNs) have emerged as a prominent class of methods within the broader field of graph representation learning [56] [41] [16]. HGNNs leverage the diversity of relationships in heterogeneous graphs to aggregate neighborhood information effectively and enrich the semantic representation of nodes. These networks often use message-passing mechanisms, which propagate and integrate information from neighboring nodes and edges based on their types and relationships. By doing so, they create representations that are not only structurally aware but also semantically rich, allowing the model to learn more meaningful and discriminative features for downstream tasks [42] [50].

One common approach within HGNNs involves the use of metapaths, predefined sequences of relations, to guide the aggregation process [3]. Metapaths enable the model to focus on specific paths that capture important semantic meaning

between node types rather than just perform a set of random walks over a given node neighborhood [27] [11]. For example, a metapath can describe *author-paper-conference (APC)* relationships in academic networks. Models like the Heterogeneous Attention Network (HAN) [41] exemplify this paradigm, employing end-to-end (E2E) training with repetitive message-passing operations during each forward-backward step, enabling effective learning on a wide variety of heterogeneous graphs. However, HAN and similar models tend to focus only on relationships between endpoint node types, which means they ignore many rich, higher-order connections that exist along the way. This narrow focus can miss important structural patterns and lead to incomplete graph representations, reducing performance on tasks that require a more holistic understanding of the network.

To overcome the limitations of traditional Heterogeneous Graph Neural Networks (HGNNs), recent advancements have focused on improving scalability while maintaining expressiveness [52] [49] [16]. A significant challenge in large-scale heterogeneous graphs is the complexity of modeling diverse relationships between different types of nodes [20], particularly when it comes to multi-hop relationships [36]. This complexity can hinder the ability of traditional models to effectively scale to larger and more semantically rich graphs. One key development in addressing this issue is the introduction of more efficient aggregation techniques that maintain the expressiveness of the model while reducing the burden of processing vast amounts of relational data.

In this paper, we present Type-Aware Heterogeneous Graph Attention Network (TAHAN), a model that addresses key challenges in heterogeneous graph learning through type-specific and type-aware aggregation. TAHAN uses the selected metapaths as routes for neighborhood construction rather than as separate aggregation channels. Neighbors reached through their prefixes are merged across metapaths and hop positions by node type, sampled, and used to learn type-specific representations that are adaptively fused. Its per-epoch attention cost is asymptotically comparable to HAN [41] when channel counts and neighborhood sizes are comparable. The one-off preprocessing cost is analyzed in Section 6. Experiments on heterogeneous graph benchmarks demonstrate competitive performance in node classification and clustering, with improvements over strong baselines on several datasets. Our main contributions are:

- Unlike approaches that aggregate separate metapath-specific representations, we use metapaths to collect neighbors of different types at both intermediate and endpoint positions. The collected neighbors are merged across the selected metapaths and neighborhood levels, then grouped by node type to form type-specific neighborhoods that support the learning of expressive, type-sensitive node representations.
- We propose a novel aggregation layer, as we aggregate from the newly constructed type-specific neighborhoods and not only from standard metapath-based neighbors. The TAHAN aggregation layer consists of two main components: a type-specific module that attends to nodes within the same multi-level neighborhood type, and a type-aware module that aggregates information across all neighborhood types, capturing the contribution of each type to the final representation. The ablation study indicates that type-aware attention contributes more to predictive performance than type-specific attention.
- We evaluate TAHAN on seven real-world heterogeneous graph datasets across node classification, clustering, and link prediction. Component ablations and attention analyses provide insight into the proposed aggregation mechanism, while sampling-budget experiments and preprocessing measurements characterize its computational behavior.

2 Related Work

The shift from homogeneous graphs to heterogeneous graphs shares common principles but also presents additional challenges, most of which concern how to combine the different heterogeneous parts of the graph to learn meaningful graph representations.

2.1 Heterogeneous Information Network Embeddings

Embedding techniques for heterogeneous information networks (HINs) aim to learn low-dimensional

representations of graphs containing multiple node and relation types. Traditional homogeneous network embeddings, such as DeepWalk [27] and node2vec [11], do not explicitly distinguish these types. Consequently, specialized HIN embedding methods incorporate type information into the representation learning process. Early methods, such as metapath2vec [3], use metapath-guided random walks to capture semantic correlations among heterogeneous nodes. The metapaths guide which typed relationships the walks follow, allowing the resulting embeddings to reflect the selected semantic context. The following subsection discusses how HGNNs incorporate this information into neighborhood aggregation and fusion.

2.2 Learning with Metapaths and Type Semantics in Heterogeneous Graphs

Neighborhood construction and aggregation. Traditional heterogeneous graph representation methods have relied heavily on metapaths to encode semantic dependencies across different node and relation types. Classical approaches including metapath2vec [3] and HIN2Vec [8] exploit metapath-guided random walks to preserve type-aware relational semantics in the learned embeddings. GNN-based models such as HAN [41], MAGNN [9], and MHNF [36] incorporate attention mechanisms to capture the contextual semantic information encoded within each metapath.

Type and relation information. More recent advances highlight that effective heterogeneous graph learning requires leveraging not only metapath-derived semantics but also richer *type-level* information. Node and edge types encode schema constraints, govern valid interaction patterns, and influence both local aggregation and global representation learning. Recent advancements have also focused on deep neural architectures, including Graph Neural Networks (GNNs) designed explicitly for HINs, such as R-GCN [30], which captures multi-relational structures through relation-specific weight matrices. GNN-based models such as SHGNN [46], RSHGNN [60], and Simple-HGNN [24] incorporate attention mechanisms and relation-specific transformations to capture type-dependent contextual information.

Fusion across metapaths and views. Building upon this trend, several recent studies expand heterogeneous learning beyond static metapath aggregation. SeHGNN [49] precomputes neighborhood aggregation and uses transformer-based semantic fusion. HeCo [23] aligns schema-level and metapath-level views through hierarchical co-contrastive learning. MV-HGNN [55] integrates multiple complementary views (structural, diffusion-based, and semantic) and fuses them via a transformer-based aggregator. GTC [37] jointly models local and global contexts using coordinated GNN and Transformer modules, while HGNN-NHSA [34] incorporates neighbor- and high-order subgraph attention to better capture higher-order dependencies.

Scalability and additional feature modalities. Recent work also emphasizes richer type semantics and scalability. BG-HGNN [33] addresses the growing number of relation types by mapping diverse relation types into a unified latent space, enabling efficient yet expressive relation-aware modeling. MM-HGNN [1] handles heterogeneous graphs with multimodal node features using modality-level attention and type-aware feature fusion, demonstrating the importance of modeling both modality and type richness. Transformer-based architectures such as EHG [40] further show that type diversity can be integrated efficiently without relying solely on metapath enumeration.

Overall, heterogeneous graph representation learning has advanced from purely metapath-guided aggregation toward more comprehensive paradigms that incorporate type-aware transformations, schema-level structures, multi-view semantic fusion, contrastive objectives, and high-order context modeling. These developments reflect a broader shift toward models that capture not only metapath-defined semantics but also the richer interplay among node types, relation types, and global heterogeneous dependencies.

Building on this evolution, our proposed method extends beyond conventional metapath-based aggregation by considering all neighbors across multiple levels of a metapath rather than restricting aggregation to only immediate or metapath-level connections. Specifically, the method groups these neighbors by their semantic types and performs structured, type-aware aggregation, enabling the model to capture more nuanced and fine-grained heterogeneous relationships. This design facilitates a deeper understanding of semantic dependencies both within and across metapaths,

thereby enhancing the expressiveness, contextual sensitivity, and overall representational power of the learned embeddings.

3 Preliminaries

We define a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$, where $\mathcal{V}$ represents the set of entities (nodes) and $\mathcal{E}$ the set of edges (relationships). $\mathcal{T}$ and $\mathcal{R}$ denote the sets of entity types and relation types, respectively. Each edge is represented as a triple $(s, r, o) \in \mathcal{E}$, where $s \in \mathcal{V}$ of type $\tau_i \in \mathcal{T}$ has a relation $r \in \mathcal{R}$ with $o \in \mathcal{V}$, which is of type $\tau_j \in \mathcal{T}$. The graph is characterized by two mapping functions: $\phi : \mathcal{V} \to \mathcal{T}$, which assigns each entity its corresponding type, and $\varphi : \mathcal{E} \to \mathcal{R}$, which maps each edge to its relation type.

Next, we formally define key concepts related to heterogeneous information networks and our specific proposed model as follows:

Definition 3.1. Metapath.

A **metapath** Φ is a sequence that represents a relationship between different types of entities in a heterogeneous graph. It is defined as: $\Phi = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where A_i denotes the entity types and R_i represents the relationships connecting these types. The metapath Φ thus captures a composite relationship $R = R_1 \circ R_2 \circ \cdots \circ R_l$ that exists between the entities A_1 and A_{l+1}, with $\circ$ indicating the composition of relationships.

Example. Consider a scenario with entities such as Authors, Papers, and Conferences (venues). A metapath like Author-Paper-Author (APA) represents co-authorship, indicating that two authors have collaborated on the same paper. Another metapath, Author-Paper-Conference-Paper-Author (APCPA), represents a relationship between authors who have published in the same conference venue. Each of these metapaths provides unique semantic insights into the relationships among authors.

Definition 3.2. Metapath-based neighbors.

The **metapath-based neighbors** $\mathcal{N}_\Phi(i)$ of a node i in a heterogeneous graph are defined as the set of nodes that are reachable from node i via the specified metapath Φ. For returning metapaths such as APA and APCPA, this includes i itself whenever a corresponding path from i back to i exists.

Example. For the metapath *Author-Paper-Author* (APA), the neighbors of author a_1 include a_1 and authors who have co-authored papers with a_1, such as a_2 and a_3. For *Author-Paper-Conference-Paper-Author* (APCPA), they include authors who have published at the same conference. These neighborhoods can be computed from products of the relation adjacency matrices along each metapath.

Definition 3.3. Heterogeneous neighborhood construction.

Given a metapath $\mathcal{P}$, the heterogeneous neighborhood $\mathcal{N}_\mathcal{P}(i)$ contains the nodes and edges reached along its prefixes from node i. For a set of metapaths $\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$, these neighborhoods are collected and their nodes are grouped by node type to form type-specific neighborhoods.

Example. Consider the following two metapaths:

$$\mathcal{P}_1 = \textit{Author-Paper-Conference-Paper-Author}\,(\textit{APCPA})$$

and:

$$\mathcal{P}_2 = \textit{Author-Paper-Author}\,(\textit{APA}).$$

For a given author a_1, the neighborhood induced by $\mathcal{P}_1$ is constructed as follows. We first enumerate all papers authored by a_1. For each such paper p_1, we identify the conference c_1 in which it was presented. From c_1, we then collect all other papers presented at the same venue and subsequently retrieve their corresponding authors. This procedure yields a heterogeneous neighborhood $\mathcal{N}_{\mathcal{P}_1}(a_1)$ composed of nodes of types *Author*, *Paper*, and *Conference*, reflecting the structural dependencies encoded by $\mathcal{P}_1$.

For the shorter metapath $\mathcal{P}_2$, we again begin by gathering all papers co-authored by a_1. From these papers, we extract all associated co-authors, resulting in the neighborhood $\mathcal{N}_{\mathcal{P}_2}(a_1)$, which spans nodes of types *Author* and *Paper*.

We merge the collected nodes by type and apply a shared sampling budget to limit neighborhood size. Nodes reached at different neighborhood levels enter the same type-specific group: for example,

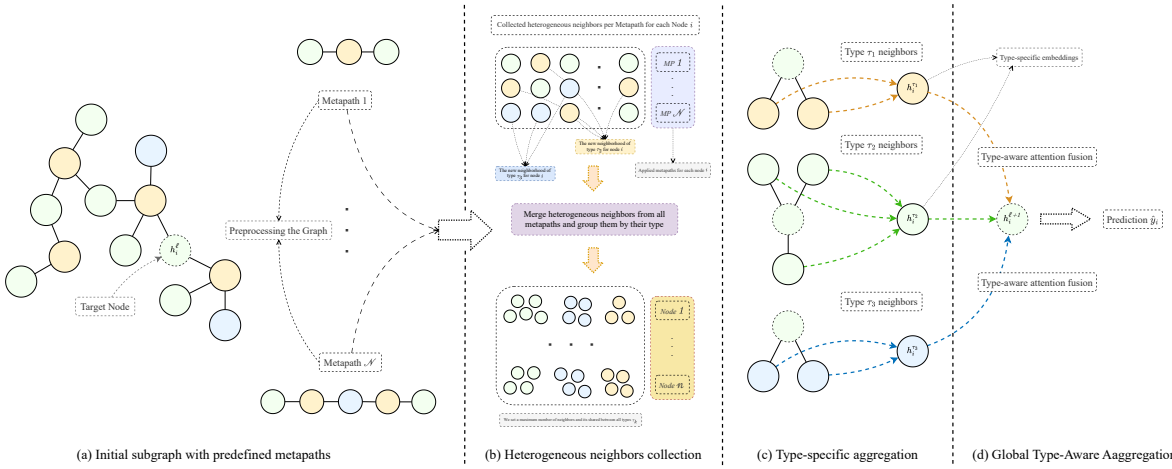


Fig. 2 Overview of TAHAN. (a) An initial heterogeneous subgraph and predefined metapaths. (b) Collection of heterogeneous neighbors along the prefixes of each selected metapath for each target node. (c) Type-specific attention aggregates neighbors within each node type. (d) Type-aware attention fuses the type-specific representations for prediction

an author's co-authors and authors who have published at the same conference belong to the Author group. Attention subsequently weights individual neighbors within that group. The construction does not explicitly encode neighborhood level. Incorporating this information is a direction for future work.

4 Proposed Method

In this section, we present the *Type-Aware Heterogeneous Graph Attention Network* (TAHAN). The overall architecture of TAHAN is illustrated in Figures 2 and 3 and includes three main components: *Heterogeneous Neighborhood Sampling*, *Type-Specific Aggregation*, and finally, what we refer to as the *Global Type-Aware Aggregation*. Our model addresses the fact that different types of entities, with distinct semantics, play unique roles in the final representation of each node. Therefore, it takes into account not only the information from nodes and the graph structure but also the hidden semantics between different types of entities, even when there are no direct links between them. Algorithm 3 outlines the overall training procedure.

4.1 Heterogeneous Neighborhood Sampling

In heterogeneous graphs, capturing meaningful information from diverse entity types and their interrelations poses a significant challenge. To address this, we introduce a *metapath-based neighborhood sampling* approach, a crucial component of our architecture. This method enables the selection of relevant neighbors by traversing predefined metapaths, encapsulating both implicit and explicit semantics among different node types. By leveraging these metapaths, the sampled neighborhoods reflect the structural properties of the graph as well as the rich semantics characterizing interactions among various entities.

We define a set of metapaths $\Phi_{\mathcal{G}}$ associated with the heterogeneous graph $\mathcal{G}$. Each metapath $\mathcal{P} \in \Phi_{\mathcal{G}}$ is represented as:

$$A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1},$$

where A_k denotes the node type at position k, and R_k represents the relationships (edges) linking these types. A node type may appear more than once in the same metapath.

To construct new adjacency matrices for the neighborhoods defined by the specified metapaths, we utilize the adjacency matrices $\mathbf{A}_{R_k}$ corresponding to each relationship R_k. For a metapath with l relationships, the adjacency matrices are constructed incrementally by performing matrix multiplications along the path. Starting with the adjacency matrix $\mathbf{A}_{R_1}$ for the first relationship, we compute each intermediate adjacency matrix through successive multiplications:

$$\mathbf{A}_{\mathcal{P},1} = \mathbf{A}_{R_1}, \quad \mathbf{A}_{\mathcal{P},2} = \mathbf{A}_{\mathcal{P},1} \cdot \mathbf{A}_{R_2}, \quad \ldots, \quad \mathbf{A}_{\mathcal{P},l} = \mathbf{A}_{\mathcal{P},l-1} \cdot \mathbf{A}_{R_l}.$$

This process results in one or more final adjacency matrices, each reflecting specific node-type relationships captured through the metapath. Finally, adjacency matrices corresponding to the same ordered pair of node types, even when reached at different hops or through different metapaths, are merged to produce a set of consolidated matrices $\{\mathbf{A}_{A_iA_j}\}$ that capture the relations embedded in the metapath.

For example, consider the metapath *Author–Paper–Author*. We begin with the *Author–Paper* adjacency matrix $\mathbf{A}_{AP}$. To compute an adjacency matrix that captures *Author–Author* relationships, we perform the multiplication $\mathbf{A}_{AA} = \mathbf{A}_{AP} \cdot \mathbf{A}_{PA}$, where $\mathbf{A}_{PA}$ represents *Paper–Author* relationships. This yields two final adjacency matrices: $\mathbf{A}_{AP}$ for *Author–Paper* and $\mathbf{A}_{AA}$ for *Author–Author*.

For a more complex example, consider the metapath *Author–Paper–Conference–Paper–Author*. We start with the *Author–Paper* adjacency matrix $\mathbf{A}_{AP}$, then compute *Author–Conference* relationships as $\mathbf{A}_{AC} = \mathbf{A}_{AP} \cdot \mathbf{A}_{PC}$, where $\mathbf{A}_{PC}$ is the *Paper–Conference* adjacency matrix. Next, we obtain an intermediate *Author–Paper* adjacency matrix $\mathbf{A}_{AP'} = \mathbf{A}_{AC} \cdot \mathbf{A}_{CP}$ using the *Conference–Paper* adjacency $\mathbf{A}_{CP}$. Finally, we derive *Author–Author* relationships as $\mathbf{A}_{AA} = \mathbf{A}_{AP'} \cdot \mathbf{A}_{PA}$. The supports of $\mathbf{A}_{AP}$ and $\mathbf{A}_{AP'}$ are merged because both connect authors to papers. This process yields three final type-pair adjacency matrices: $\mathbf{A}_{AP}$, $\mathbf{A}_{AC}$, and $\mathbf{A}_{AA}$, effectively capturing the multi-step, semantically rich relationships specified by the metapath.

Thus, for each traversal step k, the new adjacency matrix for type A_{k+1} can be expressed as:

$$\mathbf{A}_{\mathcal{P},k+1} = \mathbf{A}_{\mathcal{P},k} \cdot \mathbf{A}_{R_{k+1}}, \tag{1}$$

where $\mathbf{A}_{\mathcal{P},k}$ is the reachability matrix of the length-k prefix of $\mathcal{P}$, so that entry (u, v) counts the number of distinct path instances of that prefix joining u to v. The product is accumulated from left to right, matching the worked examples above. This iterative process continues until reaching the final node type A_{l+1}. Every prefix is retained, not only the last one: the prefix of length k is precisely the typed neighborhood of intermediate type A_{k+1}, and collecting all l prefixes is what distinguishes TAHAN from endpoint-only metapath aggregation.

Merging co-typed neighborhoods

Different metapaths, and different prefixes of the same metapath, may reach the same ordered pair of node types. Let $\mathcal{K}(\tau_i, \tau_j) = \{(\mathcal{P}, k) : \mathcal{P} \in \Phi_{\mathcal{G}},\ 1 \le k \le l_{\mathcal{P}},\ \text{type}(\mathbf{A}_{\mathcal{P},k}) = (\tau_i, \tau_j)\}$ be the set of prefixes whose endpoints carry that type pair. The *merge* operator accumulates their supports,

$$\mathbf{W}_{\tau_i\tau_j} = \sum_{(\mathcal{P},k)\in\mathcal{K}(\tau_i,\tau_j)} \mathbb{1}[\mathbf{A}_{\mathcal{P},k} > 0], \tag{2}$$

so that $(\mathbf{W}_{\tau_i\tau_j})_{uv}$ records *how many* prefixes connect u to v. The merged neighborhood of a node i with respect to type τ is then $\mathcal{N}_i^{\tau} = \{j : (\mathbf{W}_{\phi(i)\tau})_{ij} > 0\}$. Because products such as $\mathbf{A}_{AP}\mathbf{A}_{PA}$ place a non-zero on the diagonal whenever a node reaches itself through its own incident relation, $i \in \mathcal{N}_i^{\phi(i)}$ before sampling whenever such a returning path exists. This agrees with Definition 3.2 without requiring an explicit self-loop. Sampling does not, however, guarantee retention of that self-connection.

Neighbor sampling.

Merged neighborhoods can become large, increasing the cost of subsequent attention. We therefore control their expected size using the same budget $K_{\max}$ for each target node and neighbor type. Sampling uses degree-dependent Bernoulli retention rather than selecting exactly the top-K neighbors: each candidate pair is retained independently with probability

$$\begin{aligned} \Pr[(i,j) \text{ retained}] &= \min\left(1,\ \frac{K_{\max}}{d_i^{\tau}}\right), \\ d_i^{\tau} &= |\mathcal{N}_i^{\tau}|, \end{aligned} \tag{3}$$

and the retained pairs define the binary adjacency $\mathbf{A}_{\tau_i\tau_j}$ actually consumed by the attention layers. A node therefore keeps at most $K_{\max}$ neighbors per type *in expectation*, not exactly $K_{\max}$: nodes with $d_i^{\tau} \le K_{\max}$ keep their whole neighborhood, and high-degree hubs are thinned in proportion to their degree. Two consequences follow. First, the multiplicities in $\mathbf{W}$ are discarded at this point, so a neighbor reached by three prefixes is indistinguishable from one reached by a single prefix. Second, $K_{\max}$ controls the expected size of the sampled input and therefore the cost of subsequent attention. It does not limit the intermediate products

constructed before sampling. Its computational effect is analyzed in Section 6.

The overall adjacency matrix for all metapaths in $\Phi_{\mathcal{G}}$ can be defined as:

$$\mathbf{A}_{\mathcal{G}} = \bigcup_{\mathcal{P} \in \Phi_{\mathcal{G}}} \mathbf{A}_{\mathcal{P}}. \tag{4}$$

This formulation captures the heterogeneous nature of the graph, constructing adjacency matrices that reflect the diverse interactions defined by the metapaths. The resulting matrices are then used to inform further processing in the model, ensuring that both structural and semantic properties are preserved in the representation. The detailed procedure is outlined in Algorithm 1.

4.2 Type-specific Aggregation

To account for the heterogeneity among node types, we begin by applying a type-specific transformation to align the dimensionality of node features across different types. For each node i of type τ, we use a type-specific linear transformation M_τ, allowing embeddings of different types to be processed in a unified latent space. Formally, the transformed embedding h'_i for a node i is given by

$$h'_i = \mathrm{M}_\tau \cdot h_i \tag{5}$$

where h_i is the input feature vector of node i. This transformation ensures that nodes of varying types contribute information consistently in subsequent aggregation steps.

Following the transformation, we aggregate information within each type-specific neighborhood through an attention mechanism inspired by techniques used in graph attention networks (GATs) [39]. The attention score e_{ij} between a target node i and its neighbor j of type τ within the corresponding type-specific neighborhood is computed as:

$$e_{ij} = \mathrm{LeakyReLU}\left(\mathbf{a}_\tau^\top \cdot \left[h'_i \,\|\, h'_j\right]\right) \tag{6}$$

where $\mathbf{a}_\tau$ is a learnable weight vector specific to type τ, and $\|$ denotes concatenation. The attention scores are normalized across neighbors of the same type using softmax:

$$\alpha_{ij}^\tau = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i^\tau} \exp(e_{ik})} \tag{7}$$

This normalized attention weight α_{ij}^τ reflects the importance of neighbor j to target node i within its type-specific neighborhood. The aggregated embedding for node i considering only neighbors of type τ is then given by

$$h_i^\tau = \sigma\left(\sum_{j \in \mathcal{N}_i^\tau} \alpha_{ij}^\tau \cdot h'_j\right) \tag{8}$$

where σ is a non-linear activation function, such as ReLU or LeakyReLU, ensuring non-linear modeling of the aggregated neighborhood information.

Multi-head Attention: Incorporating multi-head attention into the type-specific aggregation allows the model to capture diverse aspects of neighborhood information. For each head k, we compute the attention scores and aggregate embeddings as follows:

$$h_i^\tau = \Big\Vert_{k=1}^{K} \sigma\Big(\sum_{j \in \mathcal{N}_i^\tau} \alpha_{ij}^\tau \cdot h'_j\Big) \tag{9}$$

The final output of this section depends on the metapaths and the various types of entities encountered during the sampling process. Given a set of metapaths denoted as $\Phi_{\mathcal{G}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$, we feed the node features into a type-specific attention mechanism. This process yields $|\mathcal{T}|$ groups of type-specific node embeddings for each target node i, represented as $\{h_i^{\tau_1}, \ldots, h_i^{\tau_{|\mathcal{T}|}}\}$. Each embedding $h_i^{\tau_j}$ corresponds to the representation of node i aggregated from neighbors of type τ_j, allowing us to capture the nuanced relationships and interactions specific to each entity type.

4.3 Type-aware Aggregation

Once the type-specific embeddings h_i^τ are computed for each type τ, we integrate these embeddings across types through a **type-aware aggregation** mechanism. This process captures cross-type information while preserving distinctions among types, allowing the model to leverage complementary information from different node types. The type-level attention weights are computed separately for each target node.

For each node i, we aggregate type-based embeddings using type-level attention weights.

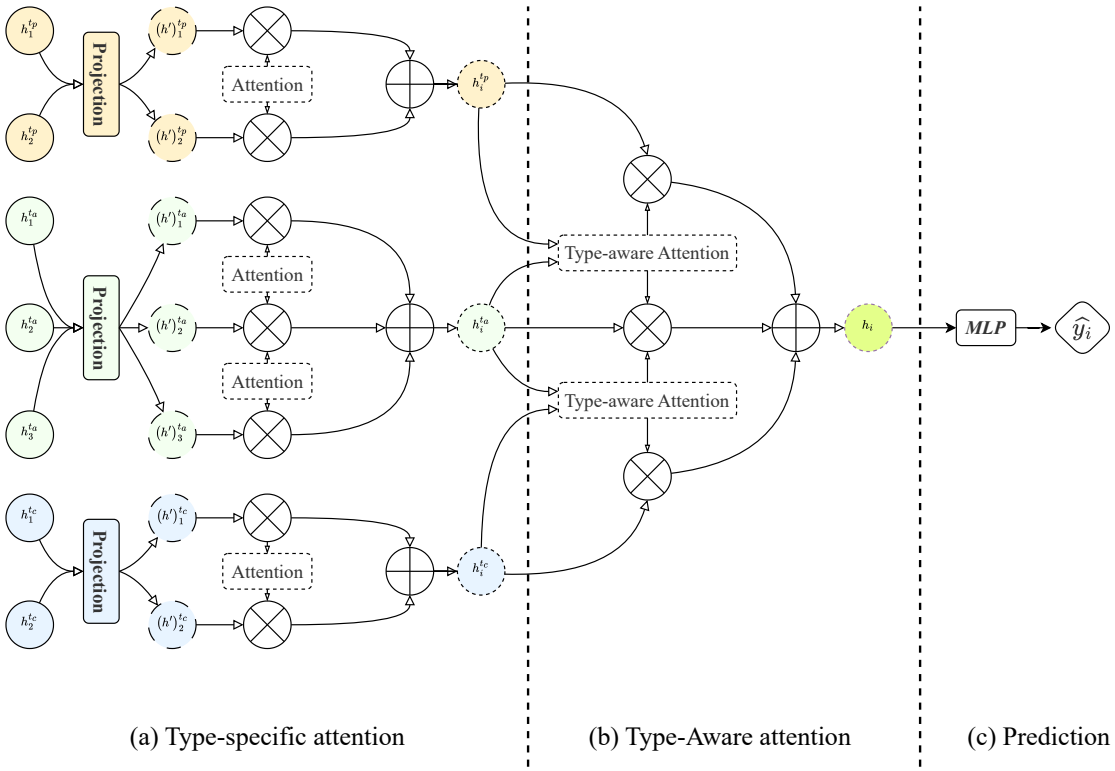


Fig. 3 TAHAN aggregation and prediction. (a) Features are projected using type-specific transformations, then aggregated through attention within each neighbor type, such as Author, Paper, or Conference. (b) Type-aware attention combines these type-specific representations into a unified embedding. (c) An MLP uses the fused embedding for node classification

To compute these weights, we calculate the type-aware attention score $e_i^{\tau_k}$ for each type embedding $h_i^{\tau_k}$ using the tanh activation function as follows:

$$e_i^{\tau_k} = \tanh\left(\mathbf{W}_\tau \cdot [h_i \,\|\, h_i^{\tau_k}] + b_\tau\right) \tag{10}$$

where $\mathbf{W}_\tau$ is a weight matrix and b_τ is a bias vector specific to type τ. The concatenation $[h_i \,\|\, h_i^{\tau_k}]$ captures the interaction between the node i and its type-specific embedding $h_i^{\tau_k}$. These scores are then normalized across types using softmax:

$$\beta_i^{\tau_k} = \frac{\exp(e_i^{\tau_k})}{\sum_{\tau \in \mathcal{T}} \exp(e_i^{\tau})} \tag{11}$$

where $\beta_i^{\tau_k}$ denotes the normalized importance of type τ_k for node i. We adopt a softmax normalization to obtain type-level attention weights. While this ensures a comparable and stable weighting across types when aggregating heterogeneous type-specific embeddings, it introduces a competitive mechanism that may suppress jointly relevant types. Exploring alternative normalization schemes, such as temperature-scaled softmax or sparse attention [25] [26], constitutes a promising direction for future work. Section 8 discusses the competition introduced by type-level softmax.

The final embedding for node i is obtained as a weighted sum of the type-specific embeddings:

$$z_i = \sum_{\tau \in \mathcal{T}} \beta_i^{\tau} \, h_i^{\tau} \tag{12}$$

Here, z_i denotes the fused output embedding, while h_i denotes the input feature used to compute the attention scores. This type-aware aggregation process allows the model to dynamically assign importance to the information from each type for a given node, effectively capturing both local (type-specific) and global (type-aware) contextual information. We can then utilize the final embeddings for various downstream tasks by designing an appropriate loss function. In our case, we focus on semi-supervised node classification and node

clustering to showcase the quality of the learned heterogeneous node embeddings. We also evaluate link prediction using the decoders described in Section 5.7.

Normalization: As stated in [24], applying L_2 normalization to the output embeddings is beneficial:

$$\tilde{z}_i = \frac{z_i}{\|z_i\|} \tag{13}$$

L_2 normalization of the output embeddings is widely adopted in retrieval-based tasks, as it ensures that the dot product becomes equivalent to cosine similarity after normalization. Interestingly, we also observe improvements in classification tasks, which aligns with findings in computer vision [28]. Furthermore, [28] suggests scaling the normalized embeddings by a factor α. We find that tuning an appropriate value for α can further enhance performance. However, the optimal value varies significantly across different datasets. For the sake of simplicity, we retain the unscaled form as shown in Equation 13.

It is worth noting that we experimented with several types of normalization, including mean and max normalization. Nonetheless, L_2 normalization consistently yielded the best results. We highlight the impact of normalization in our ablation study, as shown in Table 9.

4.4 Training Process

The training process aims to optimize the learned node representations to facilitate downstream tasks such as node classification and clustering. We employ a semi-supervised learning approach, leveraging graph information from both labeled and unlabeled nodes in the heterogeneous graph. Only labeled training nodes contribute to the classification loss. Unlabeled nodes can contribute through neighborhood aggregation. The overall objective is to minimize the classification loss, guiding the model to learn embeddings that distinguish the target classes.

For semi-supervised node classification, we can minimize cross-entropy between ground-truth labels and predictions over the labeled training nodes:

$$\mathcal{L} = -\sum_{i \in \mathcal{V}_L} \sum_{c=1}^{C} y_{i,c} \log \left[\sigma\left(f_\theta(\tilde{z}_i)\right)\right]_c \tag{14}$$

where $\mathcal{V}_L$ is the set of labeled training nodes and C is the number of classes. $\tilde{z}_i$ is the normalized final embedding of node i, f_θ is the MLP classifier with trainable parameters θ, $\sigma(\cdot)$ represents the softmax function, and $y_{i,c}$ is the label distribution for node i. The classifier maps $\tilde{z}_i$ through two fully connected hidden layers of 128 units each, with ReLU followed by dropout of 0.5 after each hidden layer. A final linear layer produces C class logits. Softmax converts these logits into the probabilities used in Equation (14). The encoder and classifier are trained jointly, with the classification loss computed only over labeled training nodes. The overall TAHAN process is summarized in Section 4.5.

For the clustering task, we employ the K-means objective after producing the final embeddings to cluster nodes. The number of clusters is set to the number of classes, and ground-truth labels are used to evaluate the resulting assignments. The objective is:

$$\mathcal{L}_{\text{clust}} = \sum_{i \in \mathcal{V}} \|\tilde{z}_i - c_{\pi(i)}\|^2 \tag{15}$$

where $\tilde{z}_i$ is the normalized final embedding of node i, $c_{\pi(i)}$ is the centroid of the cluster assigned to node i, and $\pi(i)$ denotes the cluster assignment of node i obtained from K-means.

4.5 Detailed Model Procedures

We outline the two main components of our model's learning process. First, we describe the heterogeneous neighborhood construction based on the different node types collected within a given metapath. Details of the matrix construction are provided in Algorithm 1. The sampling budget for each target node and neighbor type is treated as a tunable parameter during the construction of the neighborhood adjacency matrices. In our experiments, we evaluated multiple values for this parameter, including sampling budgets of up to 300 per target node and neighbor type. The effect of increasing this budget varies across datasets, as shown by the accuracy and training-time curves in Section 7.2. The component ablations in Section 7 keep the budget fixed to compare the attention stages and normalization under the same neighborhoods.

Section 6 analyzes the expected training-time and memory costs.

Algorithm 1: Heterogeneous neighborhood construction, merging, and sampling

Input: Heterogeneous graph $G = (V, E)$,
Metapath set $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M\}$,
Adjacency matrices
$\{\mathbf{A}_{R_1}, \mathbf{A}_{R_2}, \ldots, \mathbf{A}_{R_l}\}$,
Neighbor budget $K_{\max}$
Output: Binary type-pair adjacency matrices $\{\mathbf{A}_{\tau_i\tau_j}\}$

$\mathbf{W} \leftarrow$ empty map keyed by ordered type pairs, default $\mathbf{0}$;
foreach $\mathcal{P}_m \in \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ **do**
  $\mathbf{A}_{\text{temp}} \leftarrow \mathbf{A}_{R_1}$;
  $\tau_s \leftarrow$ head type of R_1; $\tau_t \leftarrow$ tail type of R_1;
  $\mathbf{W}[\tau_s, \tau_t] \mathrel{+}= \mathbb{1}[\mathbf{A}_{\text{temp}} > 0]$; `// length-1 prefix`
  for $k = 2$ **to** l_m **do**
    $\mathbf{A}_{\text{temp}} \leftarrow \mathbf{A}_{\text{temp}} \cdot \mathbf{A}_{R_k}$; `// left to right, Eq.` (1)
    $\tau_t \leftarrow$ tail type of R_k;
    $\mathbf{W}[\tau_s, \tau_t] \mathrel{+}= \mathbb{1}[\mathbf{A}_{\text{temp}} > 0]$; `// merge, Eq.` (2)
  end
end
foreach *type pair* (τ_i, τ_j) *present in* $\boldsymbol{W}$ **do**
  $\mathbf{A}_{\tau_i\tau_j} \leftarrow \textsc{Sample}\big(\mathbf{W}[\tau_i, \tau_j], K_{\max}\big)$; `// Algorithm 2`
end
return $\{\mathbf{A}_{\tau_i\tau_j}\}$;

Algorithm 2: SAMPLE: degree-dependent Bernoulli neighbor sampling

Input: Merged multiplicity matrix $\mathbf{W}$, budget $K_{\max}$
Output: Binary adjacency $\mathbf{A}$

$\mathcal{S} \leftarrow \{(i, j) : \mathbf{W}_{ij} > 0\}$;
$\mathbf{A} \leftarrow \mathbf{0}$;
foreach i *occurring as a row of* $\mathcal{S}$ **do**
  $d_i \leftarrow |\{j : (i, j) \in \mathcal{S}\}|$;
end
foreach $(i, j) \in \mathcal{S}$ **do**
  draw $u \sim \mathcal{U}[0, 1]$;
  if $u < \min(1, K_{\max}/d_i)$ **then** $\mathbf{A}_{ij} \leftarrow 1$;
end
return $\mathbf{A}$;

Algorithm 3: Overall training procedure for TAHAN

Input: Heterogeneous graph $G = (V, E)$,
Node features $\{h_i, \forall i \in V\}$,
Type-pair adjacencies $\{\mathbf{A}_{\tau_i\tau_j}\}$ from Algorithm 1,
Number of attention heads K,
Labeled training nodes $\mathcal{V}_L$ and their labels
Output: Final embeddings $\tilde{Z}$,
Type-specific attention weights α,
Type-aware attention weights β.

foreach *node type* $\tau \in \mathcal{T}$ **do**
  $h'_i \leftarrow \mathrm{M}_\tau \cdot h_i$ for $i \in V_\tau$; `// Eq.` (5)
end
foreach node *type* $\tau \in \mathcal{T}$ **do**
  for $k = 1$ **to** K **do**
    foreach *target node* i **do**
      Retrieve the merged neighborhood $\mathcal{N}_i^\tau = \{j : (\mathbf{A}_{\phi(i)\tau})_{ij} = 1\}$;
      foreach $j \in \mathcal{N}_i^\tau$ **do**
        $e_{ij} \leftarrow$ LeakyReLU$\big(\mathbf{a}_\tau^\top [h'_i \,\|\, h'_j]\big)$; `// Eq.` (6)
      end
      $\alpha_{ij}^\tau \leftarrow \exp(e_{ij}) / \sum_{m \in \mathcal{N}_i^\tau} \exp(e_{im})$; `// Eq.` (7)
      $h_i^{\tau,k} \leftarrow \sigma\big(\sum_{j \in \mathcal{N}_i^\tau} \alpha_{ij}^\tau \, h'_j\big)$; `// Eq.` (8)
    end
  end
  $h_i^\tau \leftarrow \Big\|_{k=1}^{K} h_i^{\tau,k}$; `// Eq.` (9)
end
foreach *target node* i **do**
  $e_i^\tau \leftarrow \tanh\big(\mathbf{W}_\tau [h_i \,\|\, h_i^\tau] + b_\tau\big)$,
  $\beta_i^\tau \leftarrow \exp(e_i^\tau) / \sum_{\tau' \in \mathcal{T}} \exp(e_i^{\tau'})$; `// Eqs.` (10), (11)
  $z_i \leftarrow \sum_{\tau \in \mathcal{T}} \beta_i^\tau \, h_i^\tau$, $\tilde{z}_i \leftarrow z_i / \|z_i\|$; `// Eqs.` (12), (13)
end
Compute class logits $f_\theta(\tilde{z}_i)$ using the MLP classifier;
Calculate cross-entropy over $i \in \mathcal{V}_L$; `// Eq.` (14)
Backpropagate and update the encoder and classifier parameters;
return $\tilde{Z}$, α, β;

Next, we present the complete learning procedure used to generate the final heterogeneous node embeddings. Details of this process are provided in Algorithm 3.

Table 1 Statistics of the benchmark datasets.

Dataset	Relations (A-B)	Number of A	Number of B	Number of A-B	Feature	#Node Types	#Classes
DBLP	Paper-Author	14328	4057	19645	334	4	4
	Paper-Conf	14328	20	14328			
	Paper-Term	14328	7723	85810			
IMDB	Movie-Actor	4278	5257	12828	3066	3	3
	Movie-Director	4278	2081	4278			
ACM	Paper-Author	3025	5959	9949	1902	4	3
	Paper-Subject	3025	56	3025			
	Paper-Term	3025	1902	255619			
AMiner	Paper-Author	6564	13329	18007	–	3	4
	Paper-Reference	6564	35890	58831			
AMiner-Large	Paper-Author	3194405	1693531	9323605	–	3	8
	Paper-Venue	3194405	3883	3194405			

Table 2 Statistics of the link prediction datasets.

Dataset	Number of Nodes	#Node Types	Number of Edges	#Relation Types	Feature	Target Relation
PubMed	63109	4	244986	10	200	Disease-Disease
LastFM	20612	3	141521	3	–	User-Artist

5 Experiments

We conduct experiments on five benchmarks for heterogeneous node-specific tasks: DBLP, IMDB, and ACM, derived from the HGB benchmark, together with AMiner, the citation graph used by the published heterogeneous baselines, and AMiner-Large, which is attribute-free and two orders of magnitude larger. Link prediction is evaluated separately on PubMed and LastFM in Section 5.7. A detailed description and the statistics of these datasets are provided in Table 1. All experiments were executed on the Toubkal supercomputer at the High Performance Computing Center (HPC) [18].

5.1 Datasets

DBLP, IMDB, ACM, and AMiner are used for the classification and clustering comparisons. AMiner-Large is evaluated separately for classification in Section 5.5, and PubMed and LastFM are used for link prediction in Section 5.7. Tables 1 and 2 summarize their statistics. The selected metapaths follow those used in prior work [9, 41].

DBLP [1]: This dataset includes 14,328 papers (P), 4,057 authors (A), 20 conferences (C), and 7,723 terms (T), with a total of 19,645 Paper-Author relationships, 14,328 Paper-Conference relationships, and 85,810 Paper-Term relationships. The author features are represented using a bag-of-words model based on keywords. For experiments, we use the following metapath set: $\{\text{APA}, \text{APCPA}\}$. The graph has 4 node types (Paper, Author, Conference, Term) and 4 classes for node classification.

IMDB [2]: This dataset includes 4,278 movies (M), 5,257 actors (A), and 2,081 directors (D), with 12,828 Movie-Actor relationships and 4,278 Movie-Director relationships. Movie features are represented using a bag-of-words model, with 3,066 feature dimensions. The graph contains 3 node types (Movie, Actor, Director) and 3 classes for node classification. The metapaths used for this dataset include Movie-Actor-Movie (MAM) and Movie-Director-Movie (MDM).

ACM [3]: This dataset contains 3,025 papers (P), 5,959 authors (A), 56 subjects (S), and 1,902 terms (T), with 9,949 Paper-Author relationships, 3,025 Paper-Subject relationships, and 255,619 Paper-Term relationships. Paper features are represented using a bag-of-words model with 1,902

[1] DBLP dataset: https://dblp.uni-trier.de/

[2] IMDB dataset: https://www.imdb.com/

[3] ACM dataset: https://www.acm.org/

feature dimensions. The graph has 4 node types (Paper, Author, Subject, Term) and 3 classes for node classification. The metapaths used are {PAP, PSP, PAPSP}.

AMiner [4]: The citation graph used by the published heterogeneous baselines [15, 23], comprising 6,564 papers (P), 13,329 authors (A) and 35,890 references (R), with 18,007 Paper-Author and 58,831 Paper-Reference relationships. The target type is Paper, with 4 classes. Node attributes are absent, so each node receives a learnable 64-dimensional input embedding from a table specific to its node type. Separate type-specific linear projections map these input embeddings into the common hidden space. We use the metapath set {PAP, PRP}. Labeled papers are split 24/6/70 following [15].

AMiner-Large [5]: The academic network released with metapath2vec [3], comprising 1,693,531 authors, 3,194,405 papers and 3,883 venues joined by 12,518,010 paper–author and paper–venue relations (25,036,020 directed edges when both directions are stored). The target type is Author, with 246,678 labeled nodes across 8 research areas. Node attributes are again absent. We use the metapath set {APA, APVPA}, which produces typed neighborhoods over Author, Paper and Venue. Compared with DBLP, IMDB, and ACM, it has approximately 190–450 times as many nodes and 58–82 times as many labeled targets. We evaluate this larger graph separately in Section 5.5, using the modified preprocessing described there, rather than including it in Table 3.

The two datasets used for the link prediction experiments of Section 5.7 are described below. Their statistics are given in Table 2.

PubMed [6]: the biomedical network of [48], with 63,109 entities of 4 types (Gene: 13,561, Disease: 20,163, Chemical: 26,522, and Species: 2,863) joined by 10 relation types and 244,986 edges. Entities were extracted with AutoPhrase, typed with bioNER and filtered by domain experts, and every node carries a 200-dimensional feature obtained by aggregating word2vec embeddings trained on the PubMed corpus. The target relation is Disease–Disease, of which 8,528 edges are held out for testing. We use the Disease-rooted metapaths {DD, DGD, DCD, DSD}, where D, G, C, and S denote Disease, Gene, Chemical, and Species, respectively.

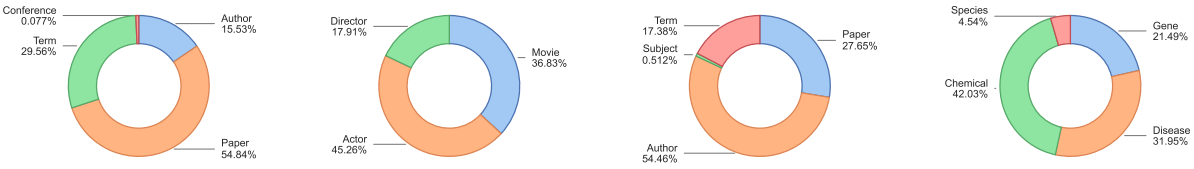
LastFM [7]: a music listening network of 20,612 nodes over 3 types (User: 1,892, Artist: 17,632, and Tag: 1,088) and 141,521 edges across 3 relation types. The target relation is User–Artist, of which 18,567 edges are held out. LastFM carries no node attributes, so each type receives a learnable embedding. The User-rooted metapath set is {UAU, UATAU, UUAU, UUATAU, UAUU}, where U denotes User, A denotes Artist, and T denotes Tag.

Figure 4 shows how node counts are distributed across types in each dataset, and Figure 5 the corresponding degree distributions. For the node classification comparison in Table 3, DBLP, IMDB, and ACM use training/validation/test proportions of 20%/20%/60%. AMiner uses 24%/6%/70%, while AMiner-Large uses 60%/20%/20% of its labeled authors. PubMed and LastFM use approximately 72%/8%/20% of the positive target-relation links. The ACM sampling-budget experiment uses a separate 60%/20%/20% split of the released labeled papers. The selection of metapaths in our approach is manually guided by those proposed in [3] and [9]. However, unlike these prior works, our method explicitly accounts for the influence of all node types along each metapath. To reduce complexity, we limit the number of metapaths considered. For example, the metapath APCPA yields three distinct sets of neighbors: one for papers, one for conferences, and one for authors. The author set includes both co-authors and authors who have published papers in the same conference, and so on.

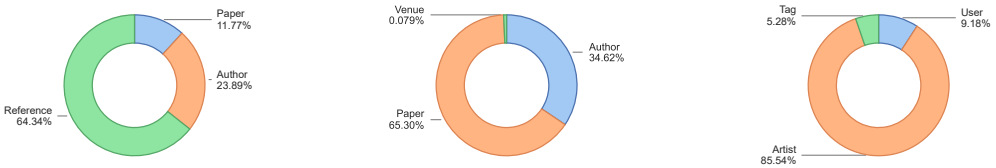
5.2 Baselines

We evaluated several state-of-the-art models across the four benchmark datasets: DBLP, IMDB, ACM and AMiner:

[4] AMiner dataset: https://www.aminer.org/
[5] AMiner dataset: https://www.aminer.org/
[6] PubMed dataset: https://github.com/yangji9181/HNE
[7] LastFM dataset: https://github.com/THUDM/HGB

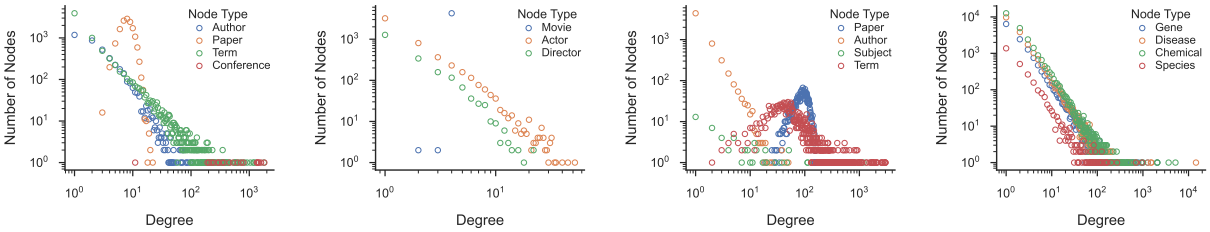


(a) DBLP **(b)** IMDB **(c)** ACM **(d)** PubMed

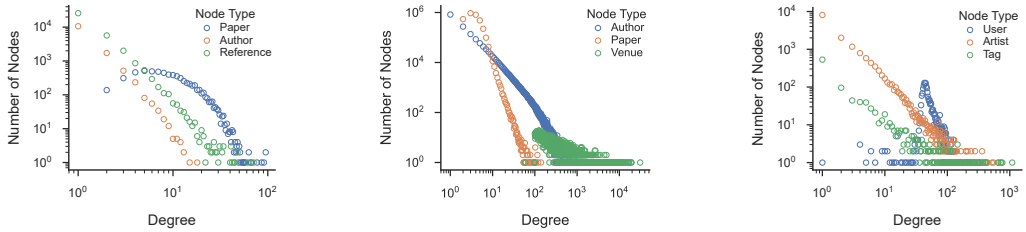


(e) AMiner **(f)** AMiner-Large **(g)** LastFM

Fig. 4 Node-type composition of DBLP, IMDB, ACM, PubMed, AMiner, AMiner-Large, and LastFM. Wedges smaller than 2.4° are displayed at 2.4° for visibility. Labels report the actual proportions



(a) DBLP **(b)** IMDB **(c)** ACM **(d)** PubMed



(e) AMiner **(f)** AMiner-Large **(g)** LastFM

Fig. 5 Degree distributions by node type in DBLP, IMDB, ACM, PubMed, AMiner, AMiner-Large, and LastFM. Degree is computed over all relation types in the released undirected graph. Both axes use logarithmic scales. Node-type colors match those in Figure 4, whose wedges use lighter tints

- **GCN** [17]: Learns node embeddings by aggregating features from immediate neighbors through graph convolutions.
- **GAT** [39]: Utilizes attention mechanisms to weigh the importance of neighboring nodes during message passing.
- **RGCN** [30]: Extends GCNs to multi-relational graphs by introducing relation-specific transformations.
- **HAN** [41]: Uses hierarchical attention to aggregate information from metapath-based heterogeneous neighborhoods.

- **GTN** [54]: Automatically learns soft selections of metapaths for message passing in heterogeneous graphs.
- **HetGNN** [56]: Integrates structural and content information from different node types using LSTM-based encoding.
- **MAGNN** [9]: Aggregates features from multiple metapaths using intra- and inter-metapath attention mechanisms.
- **HGT** [16]: A transformer-based model with type-specific attention for scalable learning on heterogeneous graphs.
- **SHGNN** [46]: Structure-Aware Heterogeneous GNN that models edge structures and semantics with attention mechanisms.
- **Simple-HGNN** [24]: A simplified and efficient heterogeneous GNN with shared weights and minimal design.
- **MHNF** [36]: Fuses multi-hop heterogeneous neighborhood information for enhanced representation learning and efficiency.
- **HeCo** [23]: Aligns schema-based and metapath-based representations via cross-view contrastive learning with view masking and hard negative sampling for robust heterogeneous node embeddings.
- **MV-HGNN** [55]: Constructs multi-view representations (including original, diffusion, and global-similarity views) and fuses them through Transformer-based aggregation to capture comprehensive semantic dependencies.
- **GTC** [37]: Employs graph transformer modules to capture contextual and high-order semantics across heterogeneous structures for improved representation learning.
- **HGNN-NHSA** [34]: Integrates neighborhood and high-order semantic attention mechanisms to adaptively aggregate information from diverse relation types in heterogeneous graphs.
- **SeHGNN** [49]: Precomputes mean neighbor aggregation and combines metapath representations through feature projection and transformer-based semantic fusion.
- **TAHAN:** The proposed Type-Aware Heterogeneous Graph Attention Network, which aggregates type-specific neighborhoods and learns attention weights across node types.

We reproduce baselines using implementations from PyTorch Geometric (PyG) [5, 6], the authors' released code, or our own reimplementations where needed. All reproduced models receive the same available node attributes, while retaining their model-specific input transformations. For datasets without node attributes, we initialize with one-hot encodings.

5.3 Hyperparameter Settings

The model hyperparameters are optimized based on their performance on the validation set. Results are averaged over 10 independent runs using different random seeds; $\pm$ denotes the standard deviation across runs. The final hyperparameters are selected according to the configuration that achieves the highest mean validation performance. Key hyperparameters in our approach include the hidden embedding dimensionality and the number of attention heads. We observed that increasing the embedding dimensionality to 512 did not yield noticeable performance gains on the DBLP and ACM datasets, so we adopted a dimensionality of 264 for these datasets. For IMDB, however, a dimensionality of 512 resulted in better performance. The number of attention heads was set to 8, determined through a grid search over five candidate values (2, 4, 6, 8, 12). The sensitivity analysis for embedding dimensionality and attention heads is illustrated in Figures 6 and 7, respectively.

The final classification layer is implemented as a multi-layer perceptron (MLP) applied to the normalized node embeddings. In our experiments, the MLP consists of two fully connected hidden layers, each with 128 units and ReLU activation, followed by a linear output layer projecting to the number of classes. Dropout with a rate of 0.5 follows the ReLU activation in each hidden layer. This is the classifier f_θ used in Equation (14). We observed that performance is relatively insensitive to the specific MLP configuration: varying the hidden layer size between 64 and 256, or using one to three hidden layers, resulted in only minor changes in accuracy. This indicates that the performance gains primarily arise from the learned node embeddings rather than extensive tuning of the classification head.

For node classification, we use Adam with a learning rate of 0.005. Weight decay is 0.001 for the input embeddings, encoder projections, type-specific attention, and MLP classifier, and zero for the type-aware fusion parameters. Attention dropout is 0.6, while the MLP dropout is 0.5 as specified above. The optimizer updates the encoder,

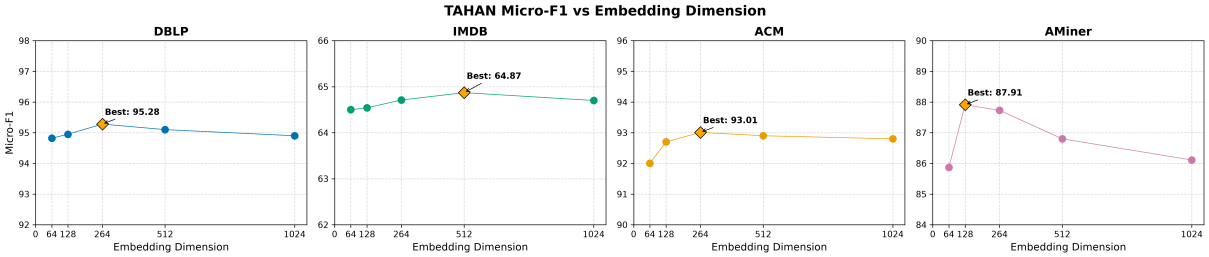


Fig. 6 Effect of embedding dimensionality on Micro-F1 across DBLP, IMDB, ACM, and AMiner. Diamond markers indicate the best score for each dataset

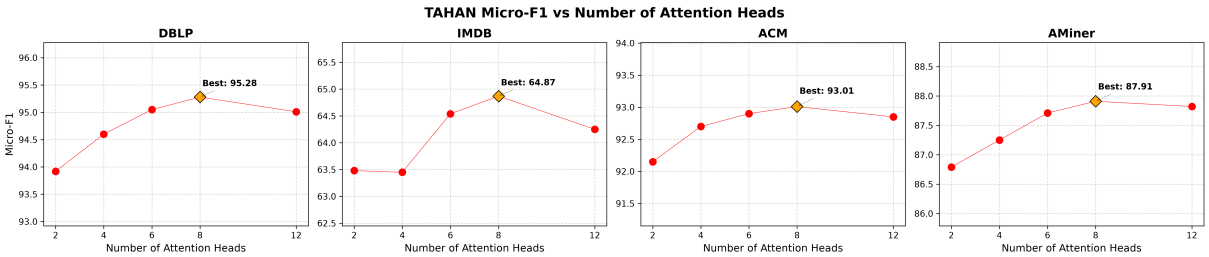


Fig. 7 Effect of the number of attention heads on Micro-F1 across DBLP, IMDB, ACM, and AMiner. Diamond markers indicate the best score for each dataset

classifier, and any learnable input embeddings jointly.

5.4 Results on Node Classification

We compare TAHAN with the baselines on the four classification benchmarks. Our classifier is trained jointly with the encoder, as described in Section 5.3. Some prior evaluations instead use a separate classifier on the learned embeddings [9, 36, 46].

The classification performance was measured using Macro-F1 and Micro-F1 metrics, which provide a balanced evaluation of the model's effectiveness across all classes and its ability to classify nodes accurately in both balanced and imbalanced datasets. Table 3 shows the results. TAHAN achieves the highest reported mean Macro-F1 and Micro-F1 on DBLP and IMDB. On ACM it ranks fourth behind SeHGNN, HGNN-NHSA, and Simple-HGNN. Section 8 discusses the variation across tasks in relation to neighborhood merging and learned fusion.

TAHAN's mean margins over the strongest baseline in each column are 0.30 and 0.16 percentage points on DBLP, 0.93 and 0.92 on IMDB, and 0.68 and 0.95 on AMiner, for Macro-F1 and Micro-F1, respectively. On ACM, its means are 0.75 and 0.86 points below SeHGNN.

Both IMDB improvements are statistically significant after Holm correction ($p_{\text{adj}} < 0.001$). On AMiner, the Macro-F1 and Micro-F1 improvements have adjusted p-values of 0.008 and 0.003, respectively.

MV-HGNN [55], HGNN-NHSA [34], and GTC [37] achieve competitive results, although SeHGNN [49] remains stronger in several comparisons. Performance therefore varies across datasets and aggregation designs.

Homogeneous GNNs also achieve results comparable to several HGNN baselines, including HAN, MAGNN, and HGT. This observation motivates examining which parts of neighborhood construction and aggregation account for the gains from heterogeneous modeling.

5.5 Scaling to AMiner-Large

For AMiner-Large, we construct reachability rows only for labeled target authors. Sparse matrix products are processed in row blocks, and degree-dependent Bernoulli sampling is applied after each metapath step. The sampled prefix supports are then merged and sampled using Equation (3). Restricting the target rows reduces the number of nodes processed, while sampling intermediate products changes the neighborhood distribution relative to Algorithm 1. With this modified construction,

Table 3 Node classification benchmark results (Macro-F1 and Micro-F1), reported as mean ± standard deviation over 10 runs. Bold and underlined values indicate the best and second-best means, respectively. A blue † indicates a significant improvement over the strongest baseline (two-sided Welch tests, Holm correction across the two metrics within each dataset, $p_{\text{adj}} < 0.05$).

Model	DBLP		IMDB		ACM		AMiner	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GCN	90.84±0.32	91.47±0.30	58.78±1.08	62.82±0.67	92.17±0.24	92.12±0.23	75.63±1.08	85.77±0.43
GAT	93.83±0.27	93.39±0.30	59.94±1.35	62.86±0.41	92.26±0.94	92.19±0.93	75.23±0.60	85.56±0.65
RGCN	91.52±0.43	92.07±0.50	58.85±0.26	62.05±0.15	91.55±0.74	91.41±0.75	63.03±2.27	82.79±1.12
HAN	91.67±0.49	92.05±0.44	61.54±0.42	62.60±0.58	90.89±0.48	90.79±0.43	63.86±2.15	82.95±1.33
GTN	93.52±0.55	93.97±0.54	60.47±0.98	63.84±0.45	91.31±0.70	91.20±0.71	72.39±1.79	84.74±1.24
HetGNN	91.76±0.43	92.33±0.41	48.25±0.67	51.16±0.65	85.91±0.25	86.05±0.28	50.06±0.90	61.49±2.50
MAGNN	93.28±0.51	93.76±0.45	60.79±2.22	63.47±1.47	90.88±0.64	90.77±0.65	71.56±1.63	83.48±1.37
HGT	93.01±0.23	93.49±0.25	60.05±1.01	60.28±0.74	91.12±0.76	91.00±0.35	70.05±1.47	82.69±1.12
MHNF	93.54±0.48	94.39±0.42	58.92±0.62	62.85±0.50	91.75±0.38	91.68±0.31	71.85±0.42	83.61±0.52
SHGNN	94.15±0.34	94.42±0.38	62.71±0.52	63.95±0.51	93.06±0.05	92.52±0.11	72.20±0.38	84.15±0.33
HeCo	92.74±0.45	93.42±0.41	63.15±0.24	62.95±0.63	89.12±0.54	88.95±0.67	71.38±1.10	78.81±1.30
GTC	92.95±0.20	93.61±0.20	63.33±0.47	63.15±0.60	89.45±0.41	89.92±0.20	72.54±0.18	82.91±0.16
MV-HGNN	94.56±0.35	94.88±0.39	63.87±0.21	63.71±0.42	93.02±0.74	92.98±0.61	74.62±0.54	85.07±0.55
HGNN-NHSA	94.35±0.40	94.67±0.42	61.32±1.05	61.45±0.95	93.61±0.38	93.47±0.48	73.45±0.84	84.68±0.78
Simple-HGNN	94.01±0.24	94.46±0.22	63.53±1.36	63.36±0.57	93.42±0.44	93.35±0.45	75.43±0.88	86.52±0.73
SeHGNN	94.75±0.50	95.12±0.20	64.02±0.24	63.85±0.31	**93.85±0.54**	**93.87±0.87**	76.83±0.57	86.96±0.64
TAHAN	**95.05±0.49**	**95.28±0.41**	**64.95±0.48**†	**64.87±0.50**†	93.10±0.45	93.01±0.43	**77.51±0.43**†	**87.91±0.47**†

Table 4 Node classification on AMiner-Large, reported as mean ± standard deviation over 10 runs. All models use the same split, featureless input, and model-selection procedure. Bold and underlined values indicate the best and second-best means, respectively. Two-sided Welch tests compare TAHAN with SeHGNN, with Holm correction across the two metrics. A blue † denotes $p_{\text{adj}} < 0.05$. $*$ marks GCN's near-majority-class performance. The majority class accounts for 27.88% of the targets.

Model	Macro-F1	Micro-F1
GCN*	5.49±2.08	28.07±1.34
HAN	92.44±1.41	92.31±1.12
Simple-HGNN	92.67±0.81	92.78±0.60
SeHGNN	92.94±0.41	92.85±0.54
TAHAN	**93.29±0.57**	**93.58±0.71**†

the final preprocessing measurement on Toubkal is 69.96 seconds at $K_{\max} = 300$, as reported in Figure 9. The 246,678 labeled authors are randomly partitioned into 60% training, 20% validation, and 20% test sets. These proportions apply to the labeled targets rather than to all authors in the graph. All models in Table 4 use the same partition within each run.

Table 4 compares five models using the same split, featureless input, and model-selection procedure, with results averaged over 10 runs. GCN operates on the type-collapsed graph, HAN uses metapath-based neighborhoods, Simple-HGNN uses relation-aware neighbor attention, and SeHGNN uses precomputed aggregation. TAHAN uses the modified construction described above, with sampling during traversal followed by the final merge-and-sample stage.

GCN's mean Micro-F1 of 28.07 is close to the majority-class share of 27.88%, indicating limited discrimination on this graph. Each venue is incident to approximately 822 papers on average, and the reported normalized adjacency has row sums as high as 87 at these hubs. Such highly connected nodes may produce similar aggregated representations, although identifying the cause of GCN's behavior requires further analysis. TAHAN exceeds the strongest baseline, SeHGNN, by 0.35 Macro-F1 and 0.73 Micro-F1 points. The Micro-F1 improvement is statistically significant after Holm correction ($p_{\text{adj}} = 0.039$). The Macro-F1 comparison gives $p_{\text{adj}} = 0.134$.

Table 5 Clustering performance using the learned embeddings, evaluated by normalized mutual information (NMI) and adjusted Rand index (ARI), reported as means over 10 runs. Bold and underlined values indicate the best and second-best results, respectively.

Model	DBLP		IMDB		ACM		AMiner	
	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
GCN	72.45	76.62	10.46	10.69	71.52	72.68	31.84	28.91
GAT	73.73	77.04	10.84	10.87	71.78	73.19	32.47	29.64
HAN	72.49	76.95	12.46	12.45	67.46	70.06	28.16	26.42
RGCN	71.05	75.21	12.12	12.21	70.89	72.14	27.53	25.71
HetGNN	69.79	75.34	9.28	10.05	41.53	34.81	21.46	26.60
MAGNN	79.63	84.58	14.41	15.22	70.48	72.32	33.21	30.37
HGT	73.21	78.34	14.94	15.69	71.26	76.62	32.84	30.02
HeCo	74.51	80.17	14.87	15.21	56.87	56.94	32.26	28.64
GTC	80.12	85.80	15.04	15.65	74.46	76.26	34.16	31.18
MV-HGNN	81.19	87.17	15.08	15.70	76.10	78.95	35.87	33.24
HGNN-NHSA	81.08	87.02	14.84	15.12	<u>77.40</u>	<u>81.10</u>	35.42	32.86
Simple-HGNN	80.84	87.12	15.05	15.67	77.32	81.04	36.12	33.57
SeHGNN	<u>81.76</u>	<u>87.23</u>	<u>15.12</u>	<u>15.72</u>	**77.65**	**81.15**	<u>36.74</u>	<u>34.21</u>
TAHAN	**83.39**	**88.14**	**16.04**	**16.38**	76.74	80.91	**37.48**	**34.96**

5.6 Results on Node Clustering

We apply K-means to the learned node embeddings, setting the number of clusters to the number of classes in each dataset. Normalized mutual information (NMI) and adjusted Rand index (ARI) measure agreement between the clusters and the ground-truth labels.

Table 5 shows that TAHAN achieves the highest NMI and ARI on DBLP, IMDB, and AMiner, indicating that its embeddings yield clusters more closely aligned with class labels than those of the evaluated baselines on these datasets. SeHGNN obtains the highest scores on ACM.

The DBLP t-SNE projections in Figure 8 provide a qualitative illustration: nodes sharing a research-area label form recognizable groups in the TAHAN panel.

5.7 Results on Link Prediction

To evaluate whether the learned embeddings encode pairwise structure, we test TAHAN on link prediction using PubMed and LastFM from the HGB suite [24]. Their statistics are summarized in Table 2.

We use the HGB evaluation protocol [24]. Test target links are excluded from the encoder's message-passing graph. Test negatives are selected from two-hop candidates, providing a harder comparison than uniformly sampled pairs. All reproduced models use the same two-hop training-negative sampler and are evaluated on the same test links and negative candidates. This differs from the random type-compatible training sampler supplied with HGB. The encoder is unchanged, and a decoder is applied to the learned embeddings: DistMult for PubMed and a dot product for LastFM.

Both link prediction experiments use Adam with a learning rate of 0.005, zero weight decay, and attention dropout of 0.3. The encoder width is 64 with eight attention heads. LastFM uses 64-dimensional learnable input embeddings. The classification MLP is not used for link prediction: the encoder and decoder are optimized using binary cross-entropy on positive training links and sampled negative links.

We report ROC-AUC and MRR using the HGB evaluator. For MRR, candidate pairs are grouped by source node and ranked by score. Each source with at least one positive candidate contributes $1/(1+\text{rank})$, where rank is the zero-based rank of its highest-ranked positive. The final score averages these contributions. Because a source can have several positive candidates, random scores can yield a substantial MRR. The random-score reference

Table 6 Link prediction performance on PubMed and LastFM (ROC-AUC and MRR), reported as mean ± standard deviation over 10 runs. Bold values indicate the best mean results. A blue † marks significant gains over the strongest baseline (two-sided Welch tests, Holm correction across the two metrics within each dataset, $p_{\text{adj}} < 0.05$).

Model	PubMed		LastFM	
	ROC-AUC	MRR	ROC-AUC	MRR
GCN	80.48±0.81	90.99±0.56	59.17±0.31	79.38±0.65
GAT	78.05±1.77	90.02±0.53	58.56±0.66	77.04±2.11
RGCN	78.29±0.18	90.26±0.24	57.21±0.09	77.68±0.17
GATNE	63.39±0.65	80.05±0.22	66.87±0.16	85.93±0.63
HetGNN	73.63±0.01	84.00±0.04	62.09±0.01	83.56±0.14
MAGNN	80.02±0.48	90.55±0.41	56.81±0.05	72.93±0.59
HGT	80.12±0.93	90.85±0.33	54.99±0.28	74.96±1.46
SeHGNN	83.95±0.27	93.02±0.31	66.59±0.62	88.61±1.25
Simple-HGNN	83.39±0.39	92.07±0.26	67.59±0.23	90.81±0.32
TAHAN	**85.18±0.77**†	**94.28±1.13**†	**69.08±0.57**†	**93.10±0.56**†

depends on the candidate counts and positive fraction, so the reported MRR should be interpreted with this evaluation setup in mind.

Table 6 reports the comparison. TAHAN exceeds the strongest listed baseline on both metrics for each dataset. On PubMed, its mean ROC-AUC and MRR exceed SeHGNN by 1.23 and 1.26 percentage points, respectively, with Holm-adjusted p-values of 0.0011 and 0.0064. On LastFM, the corresponding margins over Simple-HGNN are 1.49 and 2.29 points, with both improvements statistically significant after Holm correction ($p_{\text{adj}} < 0.001$).

6 Complexity Analysis

We analyze the computational cost of TAHAN in two stages: the offline neighborhood construction of Algorithm 1 and the attention computations during training in Algorithm 3. Throughout this section, $N = \sum_\tau |V_\tau|$ denotes the total number of nodes, $n = |V_t|$ the number of target nodes, $|\mathcal{T}|$ the number of node types reached by the selected metapaths, and $M = |\Phi_{\mathcal{G}}|$ the number of metapaths. We use $K_{\max}$ for the neighbor budget per target node and neighbor type, d for the total hidden dimensionality after concatenating attention heads, and K for the number of heads. The comparison assumes a fixed number of layers and heads. Backward propagation has the same asymptotic order as the forward pass. Baseline costs are derived from the operations described in the corresponding papers under these assumptions.

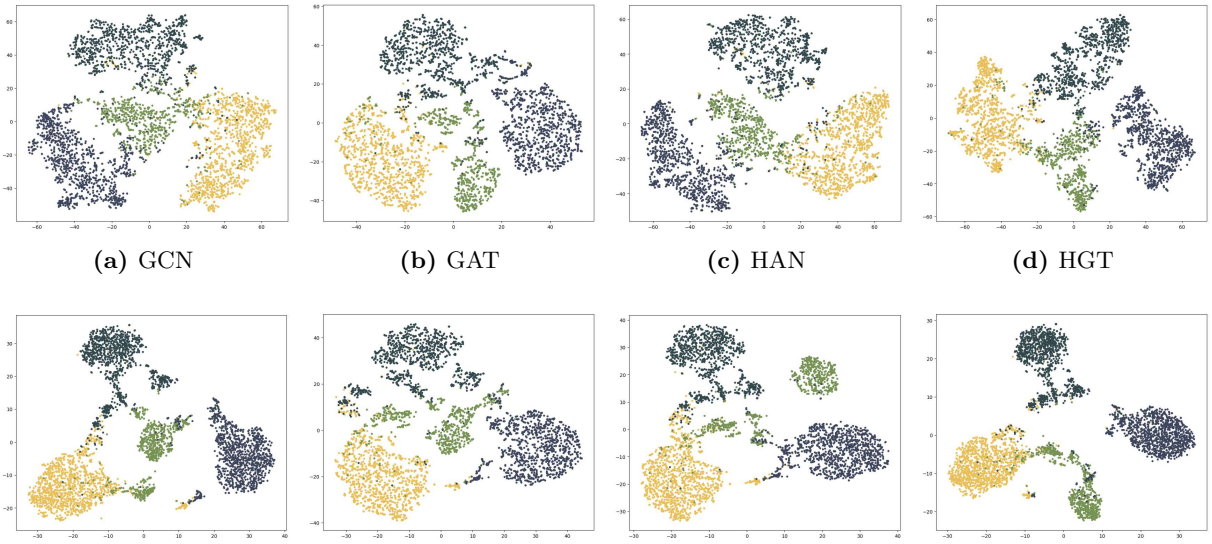
Preprocessing (Algorithm 1).

For a metapath $\mathcal{P}$ of length $l_{\mathcal{P}}$, neighborhood construction performs one sparse matrix product at each step after the first. Let $\text{nnz}(\mathbf{A})$ denote the number of nonzero entries in a matrix and Δ_{R_k} the maximum row degree of the relation matrix $\mathbf{A}_{R_k}$. A sparse product $\mathbf{A}_{\mathcal{P},k-1}\mathbf{A}_{R_k}$ then requires at most $\mathcal{O}(\text{nnz}(\mathbf{A}_{\mathcal{P},k-1})\Delta_{R_k})$ arithmetic operations. Including the collection, merging, and sampling of the prefix supports gives the upper bound

$$\mathcal{O}\Bigg(\sum_{\mathcal{P}\in\Phi_{\mathcal{G}}}\sum_{k=2}^{l_{\mathcal{P}}}\text{nnz}(\mathbf{A}_{\mathcal{P},k-1})\,\Delta_{R_k} + \sum_{\mathcal{P},k}\text{nnz}(\mathbf{A}_{\mathcal{P},k})\Bigg), \tag{16}$$

where the second sum ranges over all retained prefixes and accounts for processing their supports, including the merge in Equation (2) and the sampling pass. This preprocessing cost is incurred once for a fixed set of sampled neighborhoods, before training begins.

The main practical concern is the size of the intermediate products. Along a metapath such as APCPA, a conference can connect many papers, producing a dense author–paper prefix even when the original relation matrices are sparse. Peak preprocessing memory must accommodate both the largest intermediate product and the accumulated merged supports. It can therefore substantially exceed the size of the final sampled input. Since



(a) GCN **(b)** GAT **(c)** HAN **(d)** HGT

(e) MV-HGNN **(f)** HGNN-NHSA **(g)** SeHGNN **(h)** TAHAN

Fig. 8 t-SNE visualizations of DBLP embeddings learned by GCN, GAT, HAN, HGT, MV-HGNN, HGNN-NHSA, SeHGNN, and TAHAN. Node colors indicate research-area labels, including Artificial Intelligence, Databases, and Information Retrieval

Algorithm 1 applies sampling after constructing and merging the prefixes, $K_{\max}$ does not bound this peak memory. The modified preprocessing used for AMiner-Large, described in Section 5.5, addresses this limitation by sampling during construction. After the standard sampling procedure, the retained structure satisfies

$$\mathbb{E}\left[\sum_{\tau} \operatorname{nnz}(\mathbf{A}_{\tau_t \tau})\right] \;\le\; n\,|\mathcal{T}|\,K_{\max}, \qquad (17)$$

with equality when every target node has at least $K_{\max}$ candidate neighbors of every included type. This is an expected-size bound: independent retention does not impose a deterministic maximum on the sampled count.

Per-epoch cost (Algorithm 3).

The initial type-specific projections cost $\mathcal{O}(\sum_{\tau} |V_\tau| d^{\tau}_{\text{in}} d)$, where d^{τ}_{in} is the input feature dimension of type τ. For featureless node types, the learnable input embeddings have width d^{τ}_{in} and are followed by these same projections. Their lookup cost, $\mathcal{O}(\sum_{\tau \in \mathcal{F}} |V_\tau| d^{\tau}_{\text{in}})$, is dominated by the projection cost, where $\mathcal{F}$ denotes the featureless types. Writing $d_{\text{in}} = \max_\tau d^{\tau}_{\text{in}}$ gives the upper bound $\mathcal{O}(N d_{\text{in}} d)$. The type-specific attention stage processes each retained pair with a score computation and a weighted message, each of cost $\mathcal{O}(d)$ across all heads. By Equation (17), its expected cost is therefore $\mathcal{O}(n|\mathcal{T}|K_{\max} d)$. Here, the heads partition the total width d.

The type-aware stage uses a separate transformation for each neighbor type, as specified in Equation (10). Let r denote its output width. Each matrix has $r(d^t_{\text{in}} + d)$ entries, so evaluating these transformations costs $\mathcal{O}(n|\mathcal{T}|r(d^t_{\text{in}} + d))$. Normalization and fusion add $\mathcal{O}(n|\mathcal{T}|(r + d))$. For the comparison below, we consider input and type-attention widths of order at most d, giving the expected per-layer cost

$$\mathcal{O}\big(N d_{\text{in}} d \;+\; n|\mathcal{T}|K_{\max} d \;+\; n|\mathcal{T}|d^2\big)\,. \qquad (18)$$

Under the same width assumptions and for fixed depth and head count, the expected graph and activation storage is $\mathcal{O}(Nd + n|\mathcal{T}|K_{\max} d + n|\mathcal{T}|d)$. These terms account for projected node features, stored feature vectors for the retained pairs, and type-specific embeddings and fusion activations, respectively. Sampled indices and scalar attention scores require an additional $\mathcal{O}(n|\mathcal{T}|K_{\max})$ storage, which is dominated by the retained-pair feature term. This bound includes materialized messages rather than assuming their elimination through fused sparse aggregation. Raw input features, model parameters, and optimizer states are excluded from this storage comparison.

The encoder parameter count follows the type-specific matrices specified in Section 4:

$$\mathcal{O}\left(\sum_{\tau} d_{\text{in}}^{\tau} d + |\mathcal{T}|d + |\mathcal{T}|r(d_{\text{in}}^{t} + d)\right).$$

The terms correspond to the input projections, neighbor-attention vectors, and type-specific type-aware transformations, including their biases. When d_{in}^{t} and r are of order at most d, the last term is $\mathcal{O}(|\mathcal{T}|d^2)$. Learnable input embedding tables additionally contribute $\sum_{\tau \in \mathcal{F}} |V_\tau| d_{\text{in}}^{\tau}$ parameters. Thus, the complete model parameter count grows with the number of featureless nodes even when the graph schema and embedding widths are fixed.

For node classification, the MLP with hidden width $p = 128$ and C output classes adds $\mathcal{O}(dp+p^2+pC)$ parameters and $\mathcal{O}(n(dp+p^2+pC))$ forward computation. With p and C fixed, this does not change the asymptotic training bounds above.

Comparison with baseline models.

TAHAN and HAN [41] have a similar two-stage attention cost, but organize their channels differently: TAHAN uses node types, while HAN uses metapaths. Their neighbor-aggregation terms are $\mathcal{O}(n|\mathcal{T}|K_{\max}d)$ and $\mathcal{O}(nM\bar{K}d)$, respectively, where $\bar{K}$ is the mean number of neighbors per metapath. Comparable asymptotic costs therefore require comparable channel counts and neighborhood sizes. They do not imply equal runtime or preprocessing cost.

MAGNN [9] additionally encodes metapath instances. For mean or relational-rotation encoders, processing I instances per target node and metapath, each of length at most l, costs $\mathcal{O}(nMIld)$. TAHAN avoids this per-epoch instance-encoding term by aggregating the merged type-specific neighborhoods. In contrast, the full-batch costs of HGT [16] and Simple-HGNN [24] depend on the original edge set. TAHAN can reduce the number of pairs processed during training when $n|\mathcal{T}|K_{\max}$ is small relative to $|E|$, but this advantage depends on graph density, the sampling budget, and the cost of constructing the neighborhoods.

SeHGNN [49] precomputes neighbor aggregation and performs feature projection and semantic fusion during training. Its per-epoch expression consequently has no neighbor-aggregation term, but includes $\mathcal{O}(nM^2d)$ for attention among metapath vectors. TAHAN retains trainable neighbor attention at each epoch. Their relative training costs depend on M, $|\mathcal{T}|$, the neighborhood budget, and the feature dimensions. Neither expression establishes a universal runtime ordering. Table 7 summarizes these costs under a common set of assumptions.

For the preprocessing entries, C_{sp} and S_Φ denote the first and second sums in Equation (16), respectively. They account for sparse metapath products and processing the resulting prefix supports. For MAGNN, J_Φ counts the partial metapath instances visited during enumeration, while $nMIl$ bounds the work required to write the complete instances. With input widths of order d, independent feature propagation along the selected metapaths gives SeHGNN a preprocessing upper bound of $\mathcal{O}(Ml(N+|E|)d)$. Reusing shared prefixes can reduce this cost.

For Simple-HGNN, let e denote the relation-embedding width. Its edge-feature projection contributes $\mathcal{O}(|E|e^2)$ computation and $\mathcal{O}(|E|e)$ activation storage. Memory entries depend on the aggregation implementation: the GCN and HAN bounds use sparse aggregation without materializing a feature vector per edge; the remaining entries include the edge, instance, or semantic-attention tensors formed during their computations.

The preprocessing requirements also differ across models. GCN, HGT, and Simple-HGNN operate on the original graph and do not require metapath construction, although graph preparation or sampling can still incur setup costs. HAN constructs metapath adjacencies, whose cost depends on the intermediate sparse products. Enumerating the instances used by MAGNN requires $\mathcal{O}(nMIl)$ operations in the output size, in addition to the traversal needed to discover them. SeHGNN propagates features along normalized relation matrices during preprocessing, with a cost determined by the relation supports, feature widths, and reuse of intermediate results. Thus, the number of nonzeros in the final metapath adjacency alone does not describe the preprocessing cost of these methods.

With fixed graph schema and feature dimensions, encoder transformations and attention parameters are independent of the number of nodes and edges. GCN uses $\mathcal{O}(d^2)$ parameters per hidden layer. HGT includes node-type

Table 7 Asymptotic upper bounds for one-time preprocessing, per-epoch training time, and graph and activation memory, with fixed layer and head counts. Input dimensions and TAHAN's type-attention width r are at most of order d. TAHAN's training-time and memory bounds are in expectation over sampling. Notation and model-specific assumptions are defined in Section 6. A dash indicates no separate metapath or feature precomputation. Graph loading, adjacency preparation, raw input features, model parameters, and optimizer states are excluded

Model	Preprocessing (once)	Time / epoch	Graph/activation memory
GCN [17]	–	$\mathcal{O}(\lvert E\rvert d + Nd^2)$	$\mathcal{O}(\lvert E\rvert + Nd)$
HAN [41]	$\mathcal{O}(C_{\mathrm{sp}} + S_\Phi)$	$\mathcal{O}(Nd^2 + nMd(\bar{K} + d))$	$\mathcal{O}(Nd + nM(\bar{K} + d))$
MAGNN [9]	$\mathcal{O}(J_\Phi + nMIl)$	$\mathcal{O}(Nd^2 + nMd(Il + d))$	$\mathcal{O}(Nd + nMIld)$
HGT [16]	–	$\mathcal{O}((N + \lvert E\rvert)d^2)$	$\mathcal{O}((N + \lvert E\rvert)d)$
Simple-HGNN [24]	–	$\mathcal{O}(Nd^2 + \lvert E\rvert(d + e^2))$	$\mathcal{O}(Nd + \lvert E\rvert e)$
SeHGNN [49]	$\mathcal{O}(Ml(N + \lvert E\rvert)d)$	$\mathcal{O}(nMd(M + d))$	$\mathcal{O}(nM(d + M))$
TAHAN	$\mathcal{O}(C_{\mathrm{sp}} + S_\Phi)$	$\mathcal{O}(Nd^2 + n\lvert\mathcal{T}\rvert d(K_{\max} + d))$	$\mathcal{O}(Nd + n\lvert\mathcal{T}\rvert d(K_{\max} + 1))$

and relation-specific dense transformations, while Simple-HGNN additionally learns relation embeddings. Their counts depend on the number of node and relation types. HAN, MAGNN, and SeHGNN use metapath-specific attention, transformations, or encoders, so their parameter counts depend on M and the selected encoder architecture. TAHAN instead associates its attention parameters with the reached node types. Node-indexed learnable input embedding tables are additional parameters and retain the node-count dependence described above.

The sampling budget provides a direct trade-off between neighborhood coverage and the expected training time and memory: with the other quantities fixed, the neighbor-dependent terms grow linearly with $K_{\max}$. These costs should be considered alongside predictive performance and the one-time preprocessing cost. Section 7.2 reports accuracy and epoch time across budgets.

6.1 Neighborhood Preprocessing Time

We record the time required to construct the typed neighborhoods separately from the per-epoch training cost. Figure 9 reports the preprocessing times measured on the Toubkal supercomputer across the five node classification datasets and the two link prediction datasets, using $K_{\max} = 300$ for every dataset. The measured interval includes metapath-prefix construction, merging, and neighbor sampling. Dataset loading and the generation of negative link candidates are excluded. For LastFM, the reported time includes neighborhood construction for both the user and artist encoders.

7 Ablation Study

7.1 Variants of TAHAN Algorithm

To assess the impact of different components in TAHAN, we conducted an ablation study by evaluating several key variants, each altering a specific mechanism:

- **Variant #1:** Excludes type-specific attention, replacing it with a uniform weighting mechanism, similar to the one in [17].
- **Variant #2:** Omits type-aware attention, relying exclusively on type-specific attention for node aggregation, followed by a final layer that computes the mean of the learned type-specific embeddings.
- **Variant #3:** Removes the normalization layers from the model.

The results of this ablation study for the node classification task are summarized in Table 8 and 9 and Figure 10, with Macro-F1 and Micro-F1 scores reported for the DBLP, IMDB, and ACM datasets. The ablation results indicate that the two attention stages contribute differently to predictive performance. Replacing type-specific attention with uniform neighbor weights (Variant #1) reduces Macro-F1 and Micro-F1 by 0.38 and 0.27 points on DBLP, 1.53 and 0.82 on IMDB, and 1.28 and 0.83 on ACM. Replacing type-aware attention with mean fusion (Variant #2) produces larger reductions: 1.93 and 1.88 points on DBLP, 2.90 and 1.76 on IMDB, and 2.76 and 1.37 on ACM. Across the three datasets, the average Macro-F1 reduction is approximately 2.53 points for Variant #2 and 1.06 for Variant #1. On DBLP, the Macro-F1 reduction is about five times larger when type-aware attention is removed.

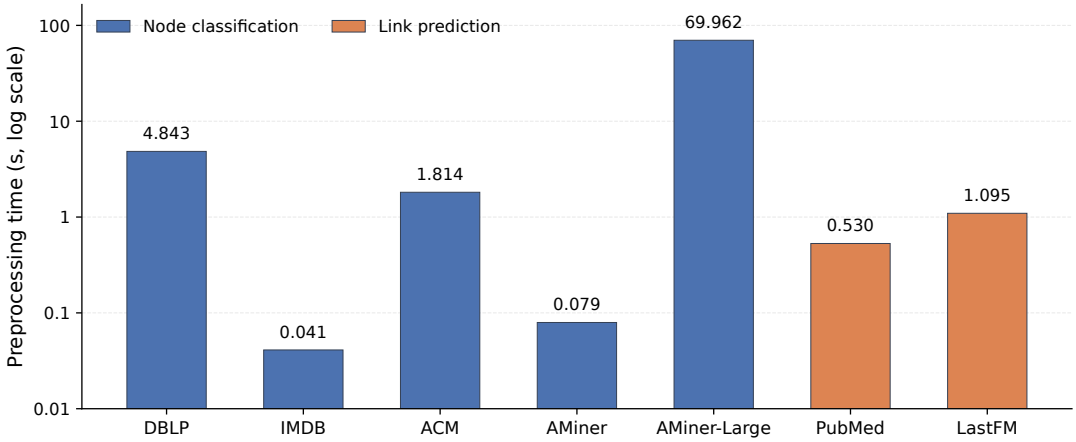


Fig. 9 Neighborhood preprocessing times for TAHAN measured on the Toubkal supercomputer with $K_{\max} = 300$ for all seven datasets. Blue bars represent node classification datasets, and orange bars represent link prediction datasets. The time axis is logarithmic, and the values above the bars are in seconds. LastFM includes neighborhood construction for both the user and artist encoders

Table 8 Node classification ablation results (Macro-F1, mean ± standard deviation over 10 runs). Variant #1 replaces type-specific attention with uniform neighbor weights. Variant #2 replaces type-aware attention with mean fusion across types. Variant #3 removes normalization.

Model Variant	DBLP	IMDB	ACM
TAHAN	**95.05±0.49**	**64.95±0.48**	**93.10±0.45**
Variant #1	94.67±0.51	63.42±0.78	91.82±0.61
Variant #2	93.12±0.47	62.05±0.80	90.34±0.58
Variant #3	93.38±0.64	62.11±0.90	90.35±0.49

Table 9 Node classification ablation results (Micro-F1, mean ± standard deviation over 10 runs). Variant #1 replaces type-specific attention with uniform neighbor weights. Variant #2 replaces type-aware attention with mean fusion across types. Variant #3 removes normalization.

Model Variant	DBLP	IMDB	ACM
TAHAN	**95.28±0.41**	**64.87±0.50**	**93.01±0.43**
Variant #1	95.01±0.10	64.05±0.24	92.18±0.48
Variant #2	93.40±0.77	63.11±0.31	91.64±0.27
Variant #3	93.88±0.84	63.57±0.35	92.03±0.92

These results support a larger predictive contribution from learned cross-type fusion than from within-type attention in the evaluated configurations. However, replacing attention also changes the trainable parameters, so the ablations do not fully distinguish learned weighting from model capacity. Variant #1 suggests that uniform aggregation within each type may retain much of TAHAN's performance with fewer attention computations. It still aggregates the retained neighbors, so it does not remove the $\mathcal{O}(n|\mathcal{T}|K_{\max}d)$ message-aggregation term in Equation (18). Its practical time and memory savings require measurement. Removing normalization (Variant #3) also reduces both metrics on all three datasets. Together, the results support retaining type-aware attention and output normalization.

7.2 Sensitivity to the Sampling Budget

The sampling budget $K_{\max}$ controls the expected number of retained neighbors per target node and neighbor type. We examine its effect

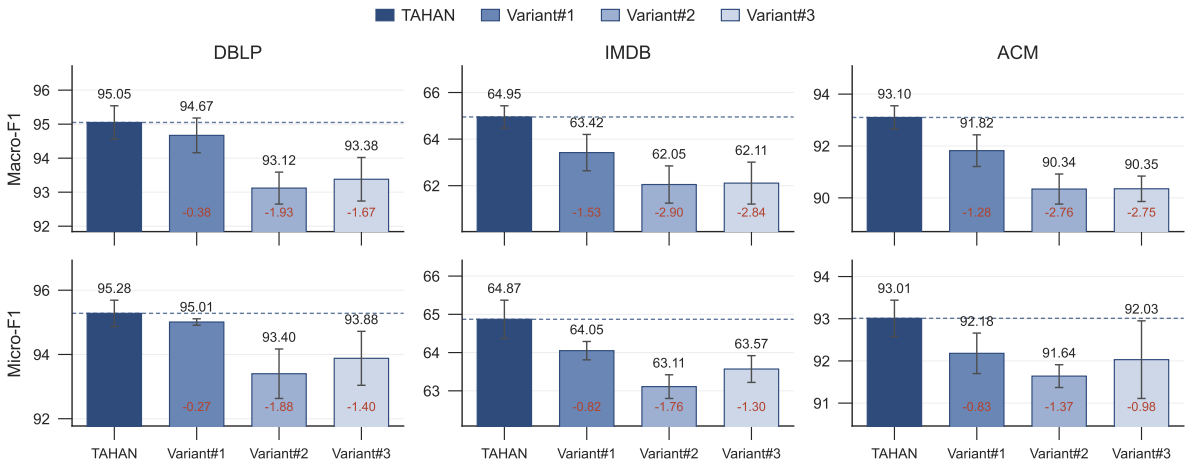


Fig. 10 Comparison of TAHAN and its ablation variants on DBLP, IMDB, and ACM. The top row reports Macro-F1 and the bottom row Micro-F1. Variant #1 uses uniform neighbor weights. Variant #2 uses mean fusion across types. Variant #3 removes normalization. Dashed lines mark the full model's scores, and red numbers show each variant's change from those scores

on neighborhood coverage, predictive performance, and training cost by varying $K_{\max}$ over $\{25, 50, 100, 150, 200, 300\}$ on DBLP, IMDB, ACM, and AMiner. We use the split protocols specified in the Datasets subsection. Within each dataset, the split, metapaths, embedding dimensions, attention heads, and optimization settings are held fixed across budgets. The reported test Micro-F1 corresponds to the checkpoint selected using validation performance. Figure 11 presents the test scores alongside the reported mean training time per epoch, with the first epoch excluded from the timing average.

The accuracy curves show that the effect of increasing the sampling budget differs across datasets. On DBLP, Micro-F1 increases from 92.48% at $K_{\max} = 25$ to 95.29% at $K_{\max} = 100$, then decreases to 94.29% at $K_{\max} = 300$. IMDB follows a similar pattern, increasing from 61.47% at $K_{\max} = 25$ to 64.87% at $K_{\max} = 150$, before reaching 64.34% at $K_{\max} = 300$. These curves show that retaining a larger neighborhood can improve predictive performance over small budgets, while further increases can provide diminishing returns.

ACM exhibits a progressive improvement across the tested budgets, increasing from 89.88% at $K_{\max} = 25$ to 93.01% at $K_{\max} = 300$. On AMiner, Micro-F1 increases from 86.34% at $K_{\max} = 25$ to 87.92% at $K_{\max} = 200$, with a small change to 87.83% at $K_{\max} = 300$. Thus, the largest observed test scores occur at different budgets across the four datasets. This supports configuring the sampling budget according to the dataset and available computational resources. The final budget is selected using validation performance.

The budget also affects the number of pairs processed by the attention layers. In DBLP, the retained pair count increases from 211,310 at $K_{\max} = 25$ to 2,439,527 at $K_{\max} = 300$. In ACM, it increases from 92,658 to 869,585 over the same range. The growth is more limited in IMDB and AMiner. IMDB retains 113,970 pairs at each budget from 150 to 300, while AMiner retains 218,239 pairs at 150 and 218,514 at 300. Increasing the budget therefore has little effect on neighborhood coverage once most candidates are already retained. These counts complement the expected training-cost analysis in Section 6. One-time neighborhood preprocessing is measured separately, as described in Section 6.1.

7.3 Analysis of the Learned Attention Values

We inspect learned attention weights in DBLP to illustrate how TAHAN weights individual neighbors within each type and combines the resulting representations across types.

We randomly selected two DBLP authors and examined their type-specific attention weights. The following examples describe the weights assigned

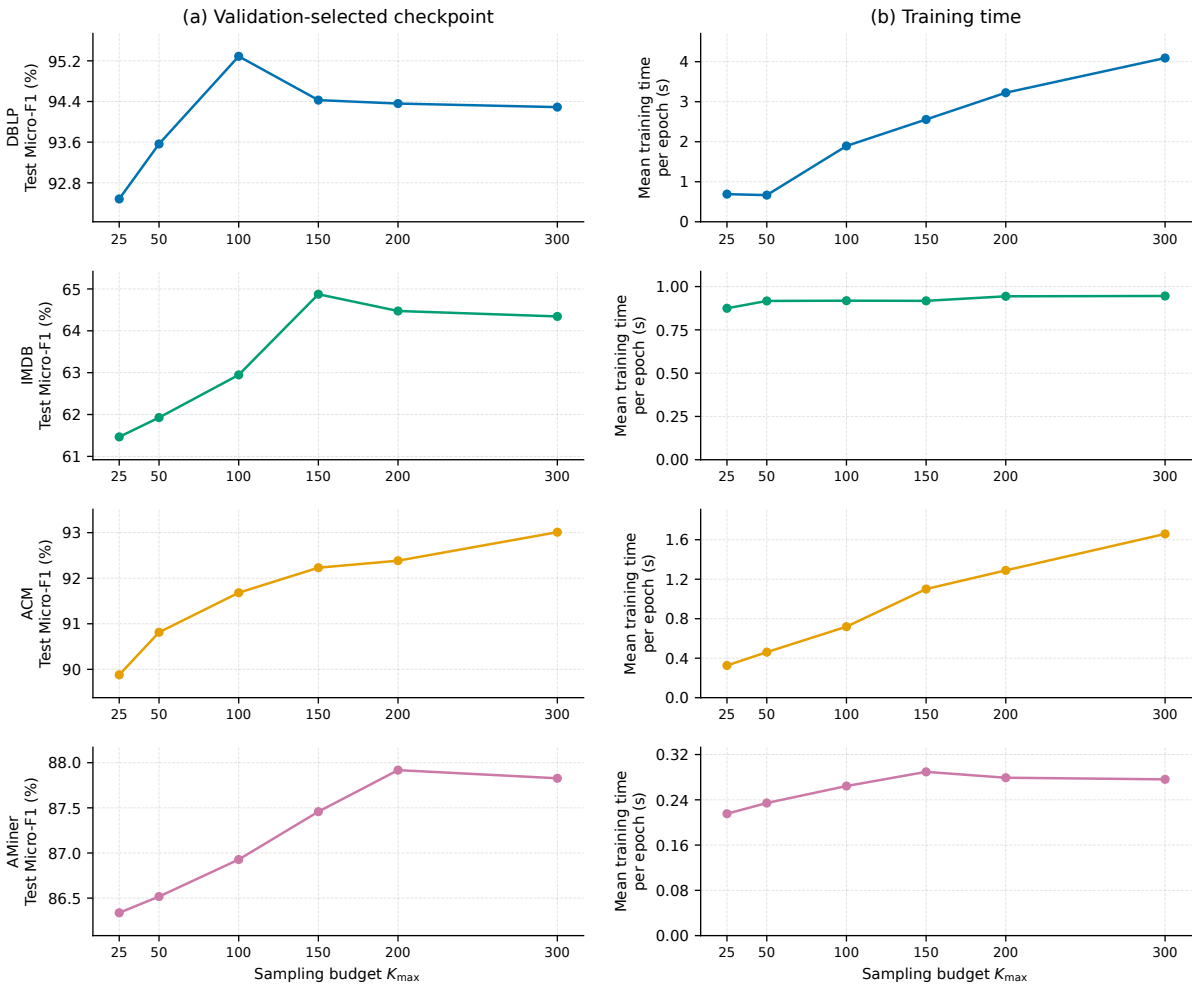


Fig. 11 Sensitivity to the sampling budget $K_{\max}$ on DBLP, IMDB, ACM, and AMiner, shown in separate rows. The left column reports test Micro-F1 at validation-selected checkpoints. The right column reports mean training time per epoch, excluding the first epoch

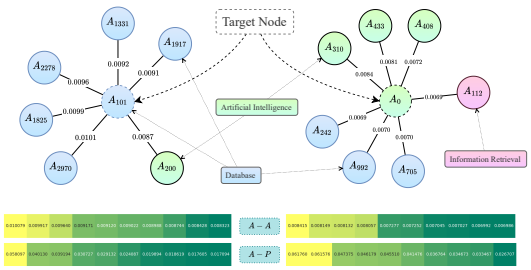


Fig. 12 Type-specific attention for authors A_{101} (left) and A_0 (right) in DBLP. The subgraphs show Author–Author neighborhoods. Edge labels give attention weights and node colors indicate research areas. The bottom heatmaps show the 10 highest-weighted neighbors within the Author–Author ($A \to A$) and Author–Paper ($A \to P$) neighborhoods of each target author

to their neighbors and the research-area labels associated with those neighbors.

Figure 12 shows the 10 highest-weighted neighbors ($k = 10$) in the Author–Author and Author–Paper neighborhoods of authors 0 and 101. Figure 13 shows the displayed Author–Author neighbors of the same targets. For these selected authors, several highly weighted neighbors share the target author's research-area label. Author 101 (A_{101}), labeled *Database*, is connected to authors A_{1331}, A_{1917}, A_{2278}, A_{1825}, and A_{2970}, all labeled *Database*, with attention weights between 0.0091 and 0.0101. Author A_{200}, labeled *Artificial Intelligence*, receives a lower weight of 0.0087.

For Author 0 (A_0), labeled *Artificial Intelligence*, the neighbors with the same label receive

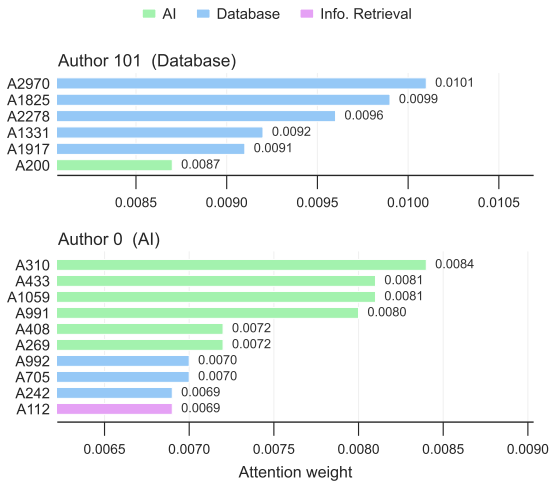


Fig. 13 Type-specific attention weights for author neighbors of A_{101} (top) and A_0 (bottom) in DBLP. The A_0 panel shows all ten author neighbors in Figure 14. Colors indicate research areas

weights between 0.0072 and 0.0084 in Figure 13. A_{310} has the highest weight among the ten displayed neighbors. Authors A_{705} and A_{992}, labeled *Database*, receive weights of 0.0070. Author A_{242}, also labeled *Database*, and author A_{112}, labeled *Information Retrieval*, each receive a weight of 0.0069. These examples show variation in attention among neighbors of the same node type.

The bottom heatmaps in Figure 12 show the 10 highest-weighted paper neighbors for A_0 and A_{101}. Their weights also vary across the Author–Paper neighborhoods. The corresponding subgraphs are omitted for clarity.

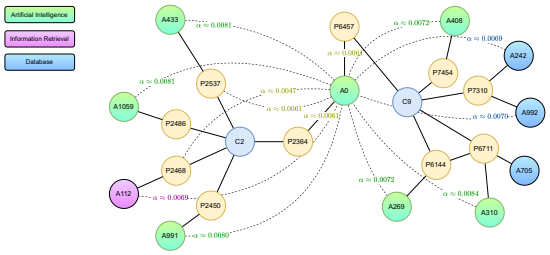


Fig. 14 Illustrative DBLP neighborhood centered on author A_0. Node IDs prefixed by A, P, and C denote authors, papers, and conferences, respectively. Author colors indicate research areas. Solid edges depict structural connections, while dashed connections show selected type-specific attention weights α from A_0 to author and paper neighbors

Figure 14 complements these attention summaries with an illustrative neighborhood centered on author A_0, showing author, paper, and conference nodes together with selected attention weights.

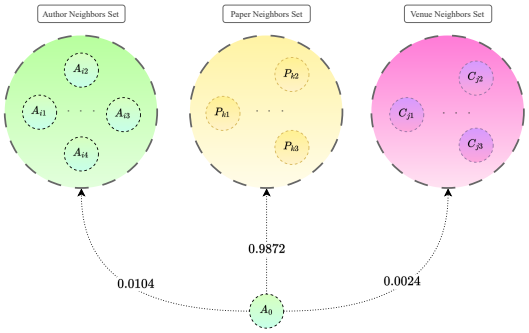


Fig. 15 Type-aware attention for author A_0 in DBLP, combining representations from Author, Paper, and Venue (Conference) neighbor sets. Edge labels give the attention weights assigned to the corresponding type-specific representations

Figure 15 shows type-aware attention for author A_0 across the Author, Paper, and Conference neighborhoods. The weights describe how their type-specific representations are combined in the fused embedding.

For this author, attention is concentrated on the Paper representation, which receives a weight of 0.9872. The Author and Conference representations receive weights of 0.0104 and 0.0024, respectively. This example illustrates a concentrated distribution across node types. Its implications for complementary type information are discussed in Section 8.

8 Discussion

Performance across tasks.

TAHAN achieves the highest reported means on three of the four classification benchmarks and three of the four clustering benchmarks (Tables 3 and 5). Its Micro-F1 margins relative to the strongest baseline range from −0.86 to 0.95 percentage points across the classification benchmarks. The link prediction results also favor TAHAN on both evaluated datasets (Table 6). These comparisons show that the constructed type-specific neighborhoods support several prediction tasks, while the variation in margins limits claims of a uniform advantage over other aggregation designs. The highest mean alone does not establish a statistically significant improvement.

Contribution of learned fusion.

The ablation study in Section 7 provides more direct evidence about the two attention stages. Across all three ablation benchmarks, replacing type-aware attention with mean fusion reduces both classification metrics more than replacing type-specific attention with uniform neighbor weights. The mean Macro-F1 reductions are 2.53 and 1.06 percentage points, respectively (Tables 8 and 9). This consistent ordering supports retaining learned fusion across types within the evaluated architecture. Because the replacements also change the trainable parameters, the results do not isolate the effect of weighting from the effect of model capacity.

Information retained by neighborhood merging.

TAHAN collects neighbors along metapath prefixes and merges those with the same node type. This construction exposes intermediate-type features to aggregation, but discards path multiplicity and the identity of the prefix through which a retained neighbor was reached. A same-type neighbor encountered at several path positions therefore contributes as a single candidate in the sampled adjacency. Type-specific attention can distinguish candidates through their features, but cannot recover discarded path information from the adjacency alone. This defines a representational limitation when the predictive signal depends on path position or on distinct semantic roles assigned to the same node type. The benchmark comparisons do not separately measure the effect of this loss.

Constraints on type-level weighting.

The two attention stages normalize over different sets: neighbors within a type in Equation (7), and type-specific representations in Equation (11). Consequently, a type's contribution to the fused embedding is not directly proportional to its number of retained neighbors. However, the type-level softmax imposes a shared weight budget, so increasing one type's weight reduces the total available to the others. The attention examples in Section 7.3 illustrate this concentration for selected nodes; they do not establish its prevalence or its effect on prediction accuracy. Testing that effect requires attention statistics over the evaluated nodes and controlled comparisons of fusion rules under matched neighborhoods, feature inputs, and model capacity.

9 Conclusion and Future Work

In this work, we introduced TAHAN, a heterogeneous graph learning framework that captures both type-specific information from each node's constructed neighborhood and global type-aware information through aggregation at the type level. Our framework models the coexistence of different node types within a neighborhood by collecting neighbors along metapath prefixes and grouping them by type across the selected paths. Type-specific attention weights individual neighbors within each group, while type-aware attention learns the relative contributions of the resulting representations to the final node embeddings. This two-stage aggregation brings together information within and across neighbor types to capture the graph's heterogeneous structure. Evaluation across seven datasets, covering node classification, clustering, and link prediction, demonstrates the applicability of the framework to citation, movie, biomedical, and music networks. The AMiner-Large experiment further extends the evaluation to a large graph without node attributes.

The ablation study highlights the importance of learning how information is combined across node types. Removing type-aware attention produces larger performance reductions than replacing type-specific attention with uniform neighbor weights across all three ablation datasets. These results identify cross-type fusion as a key component of TAHAN and motivate further exploration of simpler within-type aggregation. Building on this evidence, further interpretability studies and controlled ablation analyses can examine how learned relevance patterns, model capacity, and the layers used alongside attention contribute to performance. Such studies can deepen our understanding of attention's role and guide the design of more efficient heterogeneous graph models.

Looking ahead, several promising directions can further extend this work. One important avenue is a deeper investigation of attention mechanisms in heterogeneous and richly structured knowledge graphs, particularly examining how different attention variants behave across tasks such as link prediction and heterogeneous graph classification.

Another valuable direction involves a systematic examination of metapaths: how they are defined, selected, or discovered, and how these choices affect the information captured by the model. For TAHAN, this includes encoding path position or neighborhood level to distinguish same-type neighbors reached through different structural contexts. These directions could help streamline future models and deepen our theoretical understanding of structural semantics in heterogeneous graphs.

Statements and Declarations

Artwork preparation. The figures were prepared using draw.io, Matplotlib, and Seaborn.

Funding. No funding was received for conducting this study.

Competing interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability. The benchmark datasets are available from the sources cited and linked in Section 5.1.

References

[1] Bachiri K, Yahyaouy A, Malek M, et al (2025) Mm-hgnn: Multimodal representation learning heterogeneous graph neural network. International Journal of Computational Intelligence Systems 18. https://doi.org/10.1007/s44196-025-00820-9, URL https://api.semanticscholar.org/CorpusID:280112460

[2] Dai E, Zhao T, Zhu H, et al (2024) A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. Machine Intelligence Research 21:1011 – 1061. https://doi.org/10.1007/s11633-024-1510-8, URL https://api.semanticscholar.org/CorpusID:248239981

[3] Dong Y, Chawla N, Swami A (2017) metapath2vec: Scalable representation learning for heterogeneous networks. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining https://doi.org/10.1145/3097983.3098036, URL https://api.semanticscholar.org/CorpusID:3919301

[4] Fan W, Ma Y, Li Q, et al (2020) A graph neural network framework for social recommendations. IEEE Transactions on Knowledge and Data Engineering 34:2033–2047. https://doi.org/10.1109/tkde.2020.3008732, URL https://api.semanticscholar.org/CorpusID:225612365

[5] Fey M, Lenssen JE (2019) Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds, https://doi.org/10.48550/arXiv.1903.02428, URL https://arxiv.org/abs/1903.02428

[6] Fey M, Sunil J, Nitta A, et al (2025) PyG 2.0: Scalable learning on real world graphs. In: Temporal Graph Learning Workshop at KDD, https://doi.org/10.48550/arXiv.2507.16991, URL https://arxiv.org/abs/2507.16991

[7] Fout A, Byrd J, Shariat B, et al (2017) Protein interface prediction using graph convolutional networks. In: Neural Information Processing Systems, URL https://api.semanticscholar.org/CorpusID:26163153

[8] Fu TY, Lee WC, Lei Z (2017) Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. Proceedings of the 2017 ACM on Conference on Information and Knowledge Management https://doi.org/10.1145/3132847.3132953, URL https://api.semanticscholar.org/CorpusID:3958144

[9] Fu X, Zhang J, Meng Z, et al (2020) Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. Proceedings of The Web Conference 2020 https://doi.org/10.1145/3366423.3380297, URL https://api.semanticscholar.org/CorpusID:211031947

[10] Gilmer J, Schoenholz SS, Riley PF, et al (2017) Neural message passing for quantum chemistry. In: International Conference on Machine Learning, URL https://api.semanticscholar.org/CorpusID:9665943

[11] Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining https://doi.org/10.1145/2939672.29

39754, URL https://api.semanticscholar.org/CorpusID:207238980

[12] Gu Y, Peng S, Li Y, et al (2024) Fc-hgnn: A heterogeneous graph neural network based on brain functional connectivity for mental disorder identification. Inf Fusion 113:102619. https://doi.org/10.1016/j.inffus.2024.102619, URL https://api.semanticscholar.org/CorpusID:271751400

[13] Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Neural Information Processing Systems, URL https://api.semanticscholar.org/CorpusID:4755450

[14] Han J (2012) Mining heterogeneous information networks: the next frontier. In: Knowledge Discovery and Data Mining, https://doi.org/10.1145/2339530.2339533, URL https://api.semanticscholar.org/CorpusID:12239240

[15] He M, Wei Z, Feng S, et al (2024) Spectral heterogeneous graph convolutions via positive noncommutative polynomials. In: Proceedings of the ACM Web Conference 2024, https://doi.org/10.1145/3589334.3645515, arXiv:2305.19872

[16] Hu Z, Dong Y, Wang K, et al (2020) Heterogeneous graph transformer. Proceedings of The Web Conference 2020 https://doi.org/10.1145/3366423.3380027, URL https://api.semanticscholar.org/CorpusID:211818229

[17] Kipf T, Welling M (2016) Semi-supervised classification with graph convolutional networks. ArXiv abs/1609.02907. https://doi.org/10.48550/arXiv.1609.02907, URL https://api.semanticscholar.org/CorpusID:3144218

[18] Kissami I, Basmadjian R, Chakir O, et al (2025) Toubkal: a high-performance supercomputer powering scientific research in africa. The Journal of Supercomputing 81. https://doi.org/10.1007/s11227-025-07858-8, URL https://api.semanticscholar.org/CorpusID:281859907

[19] Lemieux GSP, Paquet E, Viktor HL, et al (2022) Geometric deep learning for protein–protein interaction predictions. IEEE Access 10:90045–90055. https://doi.org/10.1109/access.2022.3201543, URL https://api.semanticscholar.org/CorpusID:251854430

[20] Li C, Guo Z, He Q, et al (2024) Long-range meta-path search on large-scale heterogeneous graphs. In: Neural Information Processing Systems, https://doi.org/10.52202/079017-1404, URL https://api.semanticscholar.org/CorpusID:259936896

[21] Li Y, Yu R, Shahabi C, et al (2017) Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. arXiv: Learning https://doi.org/10.48550/arXiv.1707.01926, URL https://api.semanticscholar.org/CorpusID:3508727

[22] Li Z, Li Y, Luo Y, et al (2024) Graph neural networks for databases: A survey. In: International Joint Conference on Artificial Intelligence, https://doi.org/10.24963/ijcai.2024/1172, URL https://api.semanticscholar.org/CorpusID:276422118

[23] Liu N, Wang X, Han H, et al (2023) Hierarchical contrastive learning enhanced heterogeneous graph neural network. IEEE Transactions on Knowledge and Data Engineering 35(10):10884–10896. https://doi.org/10.1109/tkde.2023.3264691

[24] Lv Q, Ding M, Liu Q, et al (2021) Are we really making much progress?: Revisiting, benchmarking and refining heterogeneous graph neural networks. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining https://doi.org/10.1145/3447548.3467350, URL https://api.semanticscholar.org/CorpusID:236980351

[25] Martins AFT, Astudillo RF (2016) From softmax to sparsemax: A sparse model of attention and multi-label classification. ArXiv abs/1602.02068. https://doi.org/10.48550/arXiv.1602.02068, URL https://api.semanticscholar.org/CorpusID:16432551

[26] Niculae V, Blondel M (2017) A regularized framework for sparse and structured neural attention. ArXiv abs/1705.07704. https://doi.org/10.48550/arXiv.1705.07704, URL https://api.semanticscholar.org/CorpusID:20600085

[27] Perozzi B, Al-Rfou R, Skiena SS (2014) Deepwalk: online learning of social representations. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining https://doi.org/10.1145/2623330.2623732, URL https://api.semanticscholar.org/CorpusID:3051291

[28] Ranjan R, Castillo CD, Chellappa R (2017) L2-constrained softmax loss for discriminative face verification. ArXiv abs/1703.09507. https://doi.org/10.48550/arXiv.1703.09507, URL https://api.semanticscholar.org/CorpusID:2524204

[29] Scarselli F, Gori M, Tsoi AC, et al (2009) The graph neural network model. IEEE Transactions on Neural Networks 20:61–80. https://doi.org/10.1109/tnn.2008.2005605, URL https://api.semanticscholar.org/CorpusID:206756462

[30] Schlichtkrull M, Kipf T, Bloem P, et al (2017) Modeling relational data with graph convolutional networks. In: Extended Semantic Web Conference, https://doi.org/10.1007/978-3-319-93417-4_38, URL https://api.semanticscholar.org/CorpusID:5458500

[31] Shi C, Li Y, Zhang J, et al (2015) A survey of heterogeneous information network analysis. IEEE Transactions on Knowledge and Data Engineering 29:17–37. https://doi.org/10.1109/tkde.2016.2598561, URL https://api.semanticscholar.org/CorpusID:7885409

[32] Shi C, Hu B, Zhao WX, et al (2019) Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering 31(2):357–370. https://doi.org/10.1109/TKDE.2018.2833443

[33] Su J, Mao L, Wu C (2024) Bg-hgnn: Toward scalable and efficient heterogeneous graph neural network. ArXiv abs/2403.08207. https://doi.org/10.48550/arXiv.2403.08207, URL https://api.semanticscholar.org/CorpusID:268379398

[34] Sun W, Cheng Y, Zhao X (2025) Hgnn-nhsa: Heterogeneous graph neural network based on neighbor and high-order subgraph aggregation learning. Neurocomputing 626:129618. https://doi.org/10.1016/j.neucom.2025.129618, URL https://api.semanticscholar.org/CorpusID:276242885

[35] Sun Y, Han J (2013) Mining heterogeneous information networks: a structural analysis approach. SIGKDD Explor 14:20–28. https://doi.org/10.1145/2481244.2481248, URL https://api.semanticscholar.org/CorpusID:207204065

[36] Sun Y, Zhu D, Du H, et al (2021) Mhnf: Multi-hop heterogeneous neighborhood information fusion graph representation learning. IEEE Transactions on Knowledge and Data Engineering 35:7192–7205. https://doi.org/10.1109/tkde.2022.3186158, URL https://api.semanticscholar.org/CorpusID:235458075

[37] Sun Y, Zhu D, Wang Y, et al (2025) Gtc: Gnn-transformer co-contrastive learning for self-supervised heterogeneous graph representation. Neural networks : the official journal of the International Neural Network Society 181:106645. https://doi.org/10.1016/j.neunet.2024.106645, URL https://api.semanticscholar.org/CorpusID:268681517

[38] Tsuyuzaki K, Nikaido I (2017) Biological systems as heterogeneous information networks: A mini-review and perspectives. ArXiv abs/1712.08865. https://doi.org/10.48550/arXiv.1712.08865, URL https://api.semanticscholar.org/CorpusID:6143919

[39] Velickovic P, Cucurull G, Casanova A, et al (2017) Graph attention networks. ArXiv abs/1710.10903. https://doi.org/10.48550/arXiv.1710.10903, URL https://api.semanticscholar.org/CorpusID:3292002

[40] Wang M, Liu S, Deng Z (2025) Ehg: efficient heterogeneous graph transformer for multiclass node classification. Advances in Continuous and Discrete Models 2025. https://doi.org/10.1186/s13662-025-03885-0, URL https://api.semanticscholar.org/CorpusID:276669107

[41] Wang X, Ji H, Shi C, et al (2019) Heterogeneous graph attention network. The World Wide Web Conference https://doi.org/10.1145/3308558.3313562, URL https://api.semanticscholar.org/CorpusID:81978964

[42] Wang X, Bo D, Shi C, et al (2020) A survey on heterogeneous graph embedding: Methods, techniques, applications and sources. IEEE Transactions on Big Data 9:415–436. https://doi.org/10.1109/tbdata.2022.3177455, URL https://api.semanticscholar.org/CorpusID:227229005

[43] Wu B, Chao KM, Li Y (2024) Heterogeneous graph neural networks for fraud detection and explanation in supply chain finance. Inf Syst 121:102335. https://doi.org/10.1016/j.is.2023.102335, URL https://api.semanticscholar.org/CorpusID:266277093

[44] Wu Z, Pan S, Chen F, et al (2019) A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems 32:4–24. https://doi.org/10.1109/tnnls.2020.2978386, URL https://api.semanticscholar.org/CorpusID:57375753

[45] Xia L, Xie M, Xu Y, et al (2024) Mixrec: Heterogeneous graph collaborative filtering. Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining https://doi.org/10.1145/3701551.3703591, URL https://api.semanticscholar.org/CorpusID:274823021

[46] Xu W, Xia Y, Liu W, et al (2021) Shgnn: Structure-aware heterogeneous graph neural network. ArXiv abs/2112.06244. https://doi.org/10.48550/arXiv.2112.06244, URL https://api.semanticscholar.org/CorpusID:245124215

[47] Yang C, Pal A, Zhai A, et al (2020) Multisage: Empowering gcn with contextualized multi-embeddings on web-scale multipartite networks. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining https://doi.org/10.1145/3394486.3403293, URL https://api.semanticscholar.org/CorpusID:219557264

[48] Yang C, Xiao Y, Zhang Y, et al (2020) Heterogeneous network representation learning: A unified framework with survey and benchmark. IEEE Transactions on Knowledge and Data Engineering 34:4854–4873. https://doi.org/10.1109/tkde.2020.3045924, URL https://api.semanticscholar.org/CorpusID:229303361

[49] Yang X, Yan M, Pan S, et al (2022) Simple and efficient heterogeneous graph neural network. In: AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v37i9.26283, URL https://api.semanticscholar.org/CorpusID:250311405

[50] Ye Z, Kumar YJ, Sing GO, et al (2022) A comprehensive survey of graph neural networks for knowledge graphs. IEEE Access 10:75729–75741. https://doi.org/10.1109/access.2022.3191784, URL https://api.semanticscholar.org/CorpusID:250654689

[51] Ying R, He R, Chen K, et al (2018) Graph convolutional neural networks for web-scale recommender systems. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining https://doi.org/10.1145/3219819.3219890, URL https://api.semanticscholar.org/CorpusID:46949657

[52] Yu L, Shen J, Li J, et al (2020) Scalable graph neural networks for heterogeneous graphs. ArXiv abs/2011.09679. https://doi.org/10.48550/arXiv.2011.09679, URL https://api.semanticscholar.org/CorpusID:227054259

[53] Yue X, Wang Z, Huang J, et al (2019) Graph embedding on biomedical networks: methods, applications and evaluations. Bioinformatics 36:1241 – 1251. https://doi.org/10.1093/bioinformatics/btz718, URL https://api.semanticscholar.org/CorpusID:186206774

[54] Yun S, Jeong M, Kim R, et al (2019) Graph transformer networks. In: Neural Information Processing Systems, URL https://api.semanticscholar.org/CorpusID:202763464

[55] Zeng X, Lei F, Wang C, et al (2024) Multi-view heterogeneous graph neural networks for node classification. Data Science and Engineering 9:294 – 308. https://doi.org/10.1007/s41019-024-00253-y, URL https://api.semanticscholar.org/CorpusID:270719250

[56] Zhang C, Song D, Huang C, et al (2019) Heterogeneous graph neural network. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining https://doi.org/10.1145/3292500.3330961, URL https://api.semanticscholar.org/CorpusID:198952485

[57] Zhang J, Shi X, Xie J, et al (2018) Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In: Conference on Uncertainty in Artificial Intelligence, URL https://api.semanticscholar.org/CorpusID:3973810

[58] Zhang M, Chen Y (2018) Link prediction based on graph neural networks. In: Neural Information Processing Systems, URL https://api.semanticscholar.org/CorpusID:3573161

[59] Zhang M, Cui Z, Neumann M, et al (2018) An end-to-end deep learning architecture for graph classification. In: AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v32i1.11782, URL https://api.semanticscholar.org/CorpusID:4770492

[60] Zhu S, Zhou C, Pan S, et al (2019) Relation structure-aware heterogeneous graph neural network. 2019 IEEE International Conference on Data Mining (ICDM) pp 1534–1539. https://doi.org/10.1109/icdm.2019.00203, URL https://api.semanticscholar.org/CorpusID:207980172