Knowledge Enhanced Graph Neural Networks for Graph Completion

Luisa Werner[1,2], **Nabil Layaïda**[2], **Pierre Genevès**[1,3] and **Sarah Chlyah**[1]
[1]Institut National de Recherche en Sciences et Technologies du Numérique (INRIA)
[2]Université Grenoble Alpes (UGA)
[3]Centre National de la Recherche Scientifique (CNRS)
luisa.werner@inria.fr, nabil.layaida@inria.fr, pierre.geneves@inria.fr, sarah.chlyah@inria.fr

Abstract

Graph data is omnipresent and has a wide variety of applications, such as in natural science, social networks, or the semantic web. However, while being rich in information, graphs are often noisy and incomplete. As a result, graph completion tasks, such as node classification or link prediction, have gained attention. On one hand, neural methods, such as graph neural networks, have proven to be robust tools for learning rich representations of noisy graphs. On the other hand, symbolic methods enable exact reasoning on graphs. We propose Knowledge Enhanced Graph Neural Networks (KeGNN), a neurosymbolic framework for graph completion that combines both paradigms as it allows for the integration of prior knowledge into a graph neural network model. Essentially, KeGNN consists of a graph neural network as a base upon which knowledge enhancement layers are stacked with the goal of refining predictions with respect to prior knowledge. We instantiate KeGNN in conjunction with two state of the art graph neural networks, Graph Convolutional Networks and Graph Attention Networks, and evaluate KeGNN on multiple benchmark datasets for node classification.

1 Introduction

Graphs are ubiquitous across diverse real-world applications such as e-commerce [Liu *et al.*, 2021], natural science [Sanchez-Gonzalez *et al.*, 2018] or social networks [Wu *et al.*, 2020]. Graphs connect nodes by edges and allow to enrich them with features. This makes them a versatile and powerful data structure that encodes relational information. As graphs are often derived from noisy data, incompleteness and errors are common issues. Consequently, graph completion tasks such as node classification or link prediction have become increasingly important. These tasks are approached from different directions. In the field of deep learning, research on graph neural networks (GNNs) has gained momentum. Numerous models have been proposed for various graph topologies and applications [Ma and Tang, 2021] [Wu *et al.*, 2021] [Duan *et al.*, 2022]. The key strength of GNNs is to find meaningful representations of noisy data, that can be used for prediction tasks [Wu *et al.*, 2022]. Despite this advantage, as a subcategory of deep learning methods, GNNs are criticized for their limited interpretability and large data consumption [Susskind *et al.*, 2021]. Alongside, the research field of symbolic AI addresses the above-mentioned tasks. In symbolic AI, solutions are found by performing logic-like reasoning steps that are exact, interpretable and data-efficient. For large graphs, however, symbolic methods are often computationally expensive or even infeasible. Since techniques from deep learning and from symbolic AI have complementary pros and cons, the field of neuro-symbolic AI aims to combine both paradigms. Neuro-symbolic AI not only paves the way towards the application of AI to learning with limited data, but also allows for jointly using symbolic information (in the form of logical rules) and sub-symbolic information (in the form of real-valued data). This helps to overcome the blackbox nature of deep learning methods and to improve interpretability through symbolic representations [Susskind *et al.*, 2021].

In this paper, we present the neuro-symbolic approach **K**nowledge **e**nhanced **G**raph **N**eural **N**etworks (KeGNN) to conduct node classification given graph data and a set of prior knowledge. In KeGNN, knowledge enhancement layers are stacked on top of a GNN and adjust its predictions in order to increase the satisfaction of a set of prior knowledge. In addition to the parameters of the GNN, the knowledge enhancement layers contain learnable clause weights that reflect the impact of the prior knowledge on the predictions. Both components form an end-to-end differentiable model. KeGNN can be seen as an extension to knowledge enhanced neural networks (KENN) [Daniele and Serafini, 2022], which stack knowledge enhancement layers onto a multi-layer perceptron (MLP). However, an MLP is not powerful enough to incorporate graph structure into the representations. Thus, relational information can only be introduced by binary predicates in the symbolic part of KENN. In contrast, KeGNN is based on GNNs that process the graph structure, which makes both the neural and symbolic components sufficiently powerful to exploit the graph structure. In this work, we instantiate KeGNN in conjunction with two popular GNNs: Graph Attention Networks [Veličković *et al.*, 2018] and Graph Convolutional Networks [Kipf and Welling, 2017]. We apply KeGNN to the benchmark datasets for node classification Cora, Citeseer, PubMed [Yang *et al.*, 2016] and Flickr [Zeng *et al.*, 2020].

2 Method: KeGNN

KeGNN is a neuro-symbolic approach that can be applied to node classification tasks with the capacity of handling graph structure at the base neural network level. The model takes two types of input: (1) real-valued graph data and (2) prior knowledge expressed in first-order logic.

2.1 Graph-structured Data

A Graph $\mathbf{G} = (\mathbf{N}, \mathbf{E})$ consists of a set of n nodes $\mathbf{N}$ and a set of k edges $\mathbf{E}$ where each edge of the form (v_i, v_j) connects two nodes $v_i \in \mathbf{N}$ and $v_j \in \mathbf{N}$. The neighborhood $\mathcal{N}(v_i)$ describes the set of first-order neighbors of v_i. For an *attributed* and *labelled* graph, nodes are enriched with features and labels. Each node has a feature vector $\mathbf{x} \in \mathbb{R}^d$ of dimension d and a label vector $\mathbf{y} \in \mathbb{R}^m$. The label vector $\mathbf{y}$ contains one-hot encoded ground truth labels for m classes. In matrix notation, the features and labels of the entire graph are described as $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \mathbb{R}^{n \times m}$. A graph is *typed* if the type functions $f_{\mathbf{E}}$ and $f_{\mathbf{N}}$ assign edge types and node types to the edges and nodes, respectively. A graph with constant type functions (that assign the same edge and node type to all edges and nodes) is called *homogeneous*, whereas for *heterogeneous* graphs, nodes and edges may have different types [Ma and Tang, 2021].

Example 2.1. *A Citation Graph* $\mathbf{G}_{\text{Cit}}$ *consists of documents and citations. Fig. 1 shows an extract of the Citeseer citation graph that is used as example to guide through the method section. The documents are represented by nodes* $\mathbf{N}_{\text{Cit}}$ *and citations by edges* $\mathbf{E}_{\text{Cit}}$*. Documents can be attributed with features* $\mathbf{X}_{\text{Cit}}$ *that describe their content as Word2Vec [Adewumi* et al.*, 2020] vectors. Each node is labelled with one of six topic categories {AI, DB, HCI, IR, ML, AG}* [1] *that are encoded in* $\mathbf{Y}_{\text{Cit}}$*. Since all nodes (documents) and edges (citations) have the same type,* $\mathbf{G}_{\text{Cit}}$ *is homogeneous.*

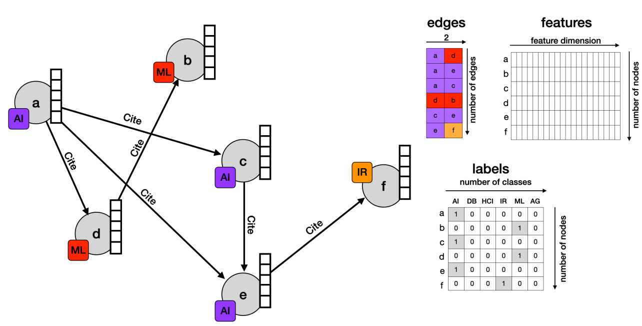

Figure (1) Example extract of the Citeseer citation graph.

2.2 Prior Knowledge

The prior knowledge $\mathcal{K}$ provided to KeGNN can be described as a set of ℓ logical clauses expressed in the logical language $\mathcal{L}$ that is defined as sets of constants $\mathcal{C}$, variables $\mathcal{X}$ and predicates $\mathcal{P}$. Predicates have an arity r of one (unary) or two (binary): $\mathcal{P} = \mathcal{P}_U \cup \mathcal{P}_B$. Predicates of arity $r > 2$ are not considered in this work. Unary predicates express properties, whereas binary predicates express relations. $\mathcal{L}$ supports the operators negation ($\neg$) and disjunction ($\vee$). Each clause $\varphi \in \mathcal{K} = \{\varphi_1, \ldots, \varphi_\ell\}$ can be formulated as a disjunction of (possibly negated) atoms $\bigvee_{j=1}^{q} o_j$ with q atoms $\{o_1, \ldots, o_q\}$. Since the prior knowledge is general, all clauses are assumed to be universally quantified. Clauses can be *grounded* by assigning constants to the free variables. A grounded clause is denoted as $\varphi[x_1, x_2, ...|c_1, c_2, ...]$ with variables $x_i \in \mathcal{X}$ and constants $c_i \in \mathcal{C}$. The set of all grounded clauses in a graph is $\mathcal{G}(\mathcal{K}, \mathcal{C})$.

Example 2.2. *The graph* $\mathbf{G}_{\text{Cit}}$ *in Fig. 1 can be expressed in* $\mathcal{L}$*. Nodes are represented by a set of constants* $\mathcal{C} = \{a, b, \ldots, f\}$*. Node labels are expressed as a set of unary predicates* $\mathcal{P}_U = \{\text{AI}, \text{DB}, \ldots, \text{AG}\}$ *and edges as a set of binary predicates* $\mathcal{P}_B = \{\text{Cite}\}$*.* $\mathcal{L}$ *has a set of variables* $\mathcal{X} = \{x, y\}$*. The atom* AI(x) *expresses the membership of* x *to the class* AI *and* Cite(x, y) *expresses the existence of a citation between* x *and* y*. A set of prior knowledge* $\mathcal{K}$ *can be written as* $\ell = 6$ *disjunctive clauses in* $\mathcal{L}$*. Here, the assumption is denoted that two papers that cite each other have the same document class:*

$$\forall xy \neg \text{AI(x)} \vee \neg \text{Cite(x, y)} \vee \text{AI(y)}$$
$$\forall xy \neg \text{DB(x)} \vee \neg \text{Cite(x, y)} \vee \text{DB(y)}$$
$$\ldots$$

The atoms are grounded by replacing the variables x *and* y *with the constants* $\{a, b, \ldots f\}$ *to obtain sets of unary groundings* $\{\text{AI(a)}, \text{ML(b)}, \ldots, \text{IR(f)}\}$ *and binary groundings* $\{\text{Cite(a, d)}, \text{Cite(a, e)}, \ldots, \text{Cite(a, f)}\}$*. Assuming a closed world and exclusive classes, other facts could be derived, such as* $\{\neg\text{DB(a)}, \neg\text{IR(a)}, \ldots, \neg\text{Cite(a, b)}\}$*. For the sake of simplicity, these are omitted here.*

2.3 Node Classification

Node classification is a subtask of knowledge graph completion on a graph $\mathbf{G}$ with the objective to assign classes to nodes where they are unknown. This task is accomplished given node features $\mathbf{X}$, edges $\mathbf{E}$ and some prior knowledge $\mathcal{K}$ encoded as a set of clauses in $\mathcal{L}$. A predictive model is trained on a subset of the graph $\mathbf{G}_{\text{train}}$ with ground truth labels $\mathbf{Y}_{\text{train}}$ and validated on a test set $\mathbf{G}_{\text{test}}$ for which the ground truth labels are compared to the predictions in order to assess the predictive performance. Node classification can be studied in a *transductive* or *inductive* setting. In a transductive setting, the entire graph is available for training, but the true labels of the test nodes are masked. In an inductive setting, only the nodes in the training set and the edges connecting them are available, making it more challenging to classify unseen nodes.

2.4 Fuzzy Semantics

Let us consider an attributed and labelled graph $\mathbf{G}$ and a set of prior knowledge $\mathcal{K}$. While $\mathcal{K}$ can be defined in the logic language $\mathcal{L}$, the neural component in KeGNN relies on continuous and differentiable representations. To interpret Boolean

[1]The classes are abbreviations for the categories *Artificial Intelligence, Databases, Human-Computer Interaction, Information Retrieval, Machine Learning and Agents*.

logic in the real-valued domain, KeGNN uses fuzzy logic [Zadeh, 1988], which maps Boolean truth values to the continuous interval $[0, 1] \subset \mathbb{R}$. A constant in $\mathcal{C}$ is interpreted as a real-valued feature vector $\mathbf{x} \in \mathbb{R}^d$. A predicate in $\mathcal{P}$ with arity r is interpreted as a function $f_{\mathcal{P}} : \mathbb{R}^{r \times d} \mapsto [0, 1]$ that takes r feature vectors as input and returns a truth value.

Example 2.3. *In the example, a unary predicate $P_U \in \mathcal{P}_U = \{\mathrm{AI}, \mathrm{DB}, \ldots\}$ is interpreted as a function $f_{P_U} : \mathbb{R}^d \mapsto [0, 1]$ that takes a feature vector $\mathbf{x}$ and returns a truth value indicating whether the node belongs to the class encoded as P_U. The binary predicate $\mathrm{Cite} \in \mathcal{P}_B$ is interpreted as the function*

$$f_{\mathrm{Cite}}(v_i, v_j) = \begin{cases} 1, & \textit{if } (v_i, v_j) \in \mathbf{E}_{\mathrm{Cit}} \\ 0, & \textit{else.} \end{cases}$$

f_{Cite} returns 1 (true) if there is an edge between two nodes v_i and v_j in $\mathbf{G}_{\mathrm{Cit}}$ and 0 otherwise.

T-conorm functions $\perp : [0, 1] \times [0, 1] \mapsto [0, 1]$ [Klement *et al.*, 2013] take real-valued truth values of two literals[2] and define the truth value of their disjunction. The Gödel t-conorm function for two truth values $\mathbf{t}_i, \mathbf{t}_j$ is defined as

$$\perp(\mathbf{t}_i, \mathbf{t}_j) \mapsto \max(\mathbf{t}_i, \mathbf{t}_j).$$

To obtain the truth value of a clause $\varphi : o_1 \vee ... \vee o_q$, the function $\perp$ is extended to a vector $\mathbf{t}$ of q truth values: $\perp(\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_q) = \perp(\mathbf{t}_1, \perp(\mathbf{t}_2 ... \perp(\mathbf{t}_{q-1}, \mathbf{t}_q)))$. Fuzzy negation over truth values is defined as $\mathbf{t} \mapsto 1 - \mathbf{t}$ [Zadeh, 1988].

Example 2.4. *Given the clause $\varphi_{\mathrm{AI}} : \forall xy\ \neg\mathrm{AI(x)} \vee \neg\mathrm{Cite(x,y)} \vee \mathrm{AI(y)}$ and its grounding $\varphi_{\mathrm{AI}}[x, y|a, b] : \mathrm{AI(a)} \vee \neg\mathrm{Cite(a,b)} \vee \mathrm{AI(b)}$ to the constants a and b and truth values for the grounded predicates $\mathrm{AI(a)} = \mathbf{t}_1$, $\mathrm{AI(b)} = \mathbf{t}_2$ and $\mathrm{Cite(a,b)} = \mathbf{t}_3$, the truth value of $\varphi_{\mathrm{AI}}[x, y|a, b]$ is $\max\{\max\{(1 - \mathbf{t}_1), (1 - \mathbf{t}_3)\}, \mathbf{t}_2\}$.*

2.5 Model Architecture

The way KeGNN computes the final predictions can be divided in two stages. First, a GNN predicts the node classes given the features and the edges. Subsequently, the knowledge enhancement layers use the predictions as truth values for the grounded unary predicates and update them with respect to the knowledge. An overview of KeGNN is given in Fig. 2.

Neural Component

The role of the GNN in the neural component is to exploit feature information in the graph structure. The key strength of a GNN is to enrich node representations with graph structure by nesting k message passing layers [Wu *et al.*, 2022]. Per layer, the representations of neighboring nodes are aggregated and combined to obtain updated representations. The node representation v_i^{k+1} in the k-th message passing layer is

$$v_i^{k+1} = \mathrm{combine}\big(v_i^k, \mathrm{aggregate}\big(\{v_j^k | v_j^k \in \mathcal{N}(v_i)\}\big)\big).$$

The layers contain learnable parameters that are optimized with backpropagation. In this work, we consider two well-known GNNs as components for KeGNN: Graph Convolutional Networks (GCN) [Kipf and Welling, 2017] and Graph Attention Networks (GAT) [Veličković *et al.*, 2018]. While GCN considers the graph structure as given, GAT allows for assessing the importance of the neighbors with attention weights α_{ij} between node v_i and node v_j. In case of multi-head attention, the attention weights are calculated multiple times and concatenated which allows for capturing different aspects of the input data. In KeGNN, the GNN implements the functions f_{P_U} (see Section 2.4). In other words, the predictions are used as truth values for the grounded unary predicates in the symbolic component.

Symbolic Component

To refine the predictions of the GNN, one or more knowledge enhancement layers are stacked onto the GNN to update its predictions $\mathbf{Y}$ to $\mathbf{Y}'$. The goal is to increase the satisfaction of the prior knowledge. The predictions $\mathbf{Y}$ of the GNN serve as input to the symbolic component where they are interpreted as fuzzy truth values for the unary grounded predicates $\mathbf{U} := \mathbf{Y}$ with $\mathbf{U} \in \mathbb{R}^{n \times m}$. Fuzzy truth values for the groundings of binary predicates are encoded as a matrix $\mathbf{B}$ where each row represents an edge (v_i, v_j) and each column represents an edge type e. In the context of node classification, the GNN returns only predictions for the node classes, while the edges are assumed to be given. A binary grounded predicate is therefore set to truth value 1 (true) if an edge between two nodes v_i and v_j exists:

$$\mathbf{B}_{[(v_i, v_j), e]} = \begin{cases} 1, & \text{if } (v_i, v_j) \text{ of type } e \in \mathbf{E} \\ 0, & \text{else.} \end{cases}$$

Example 2.5. *In case of the beforementioned citation graph of Fig. 1, $\mathbf{U}$ and $\mathbf{B}$ are defined as:*

$$\mathbf{U} := \begin{bmatrix} \mathrm{AI(a)} & \ldots & \mathrm{AG(a)} \\ \mathrm{AI(b)} & \ldots & \mathrm{AG(b)} \\ \vdots & & \vdots \\ \mathrm{AI(f)} & \ldots & \mathrm{AG(f)} \end{bmatrix} \quad \mathbf{B} := \begin{bmatrix} \mathrm{Cite(a,d)} \\ \mathrm{Cite(a,e)} \\ \mathrm{Cite(a,c)} \\ \vdots \\ \mathrm{Cite(c,e)} \\ \mathrm{Cite(e,f)} \end{bmatrix}$$

To enhance the satisfaction of clauses that contain both unary and binary predicates, their groundings are joined into one matrix $\mathbf{M} \in \mathbb{R}^{k \times P}$ with $P = 2 \cdot |\mathcal{P}_U| + \mathcal{P}_B|$. $\mathbf{M}$ is computed by joining $\mathbf{U}$ and $\mathbf{B}$ so that each row of $\mathbf{M}$ represents an edge (v_i, v_j). As a result, $\mathbf{M}$ contains all required grounded unary predicates for v_i and v_j.

Example 2.6. *For the example citation graph, we obtain $\mathbf{M}$ as follows:*

$$\mathbf{M} = \begin{array}{c} \begin{array}{ccc} \overbrace{\hspace{5cm}}^{\text{unary grounded predicates for x}} & \overbrace{\hspace{5cm}}^{\text{unary grounded predicates for y}} & \overbrace{\hspace{2cm}}^{\text{binary groundings}} \end{array} \\ \left[\begin{array}{cccc|cccc|c} \mathrm{AI(a)} & \mathrm{DB(a)} & \ldots & \mathrm{AG(a)} & \mathrm{AI(d)} & \mathrm{DB(d)} & \ldots & \mathrm{AG(d)} & \mathrm{Cite(a,d)} \\ \mathrm{AI(a)} & \mathrm{DB(a)} & \ldots & \mathrm{AG(a)} & \mathrm{AI(e)} & \mathrm{DB(e)} & \ldots & \mathrm{AG(e)} & \mathrm{Cite(a,e)} \\ \mathrm{AI(a)} & \mathrm{DB(a)} & \ldots & \mathrm{AG(a)} & \mathrm{AI(c)} & \mathrm{DB(c)} & \ldots & \mathrm{AG(c)} & \mathrm{Cite(a,c)} \\ \vdots & & & \vdots & \vdots & & & \vdots & \vdots \\ \mathrm{AI(c)} & \mathrm{DB(c)} & \ldots & \mathrm{AG(c)} & \mathrm{AI(e)} & \mathrm{DB(e)} & \ldots & \mathrm{AG(e)} & \mathrm{Cite(c,e)} \\ \mathrm{AI(e)} & \mathrm{DB(e)} & \ldots & \mathrm{AG(e)} & \mathrm{AI(f)} & \mathrm{DB(f)} & \ldots & \mathrm{AG(f)} & \mathrm{Cite(e,f)} \end{array}\right] \end{array}$$

For each clause $\varphi \in \mathcal{K}$, a *clause enhancer* is instantiated. Its aim is to compute updates $\delta\mathbf{M}_\varphi$ for the groundings in $\mathbf{M}$

[2] A literal is a (possibly negated) grounded atom, e.g. $\mathrm{AI(a)}$

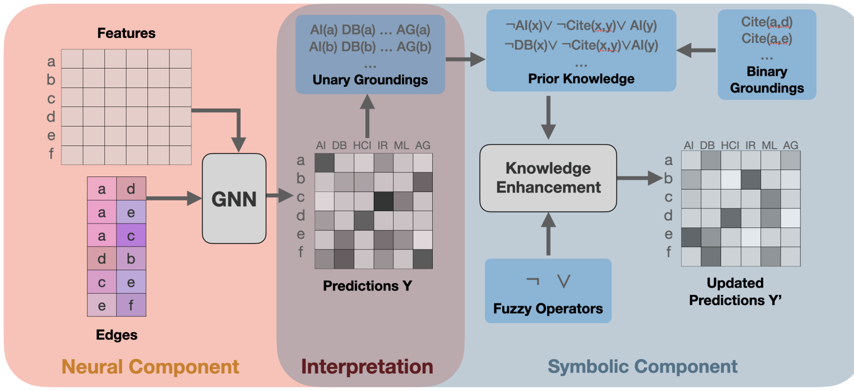

Figure (2) Overview of KeGNN.

that increase the satisfaction of φ. First, fuzzy negation is applied to the columns of $\mathbf{M}$ that correspond to negated atoms in φ. Then $\delta\mathbf{M}_\varphi$ is computed by a *t-conorm boost function* ϕ [Daniele and Serafini, 2020]. $\phi : [0,1]^q \mapsto [0,1]^q$ takes q truth values and returns changes to those truth values such that $\bot(\mathbf{t}) \leq \bot(\mathbf{t} + \phi(\mathbf{t}))$. [Daniele and Serafini, 2020] propose the following differentiable t-conorm boost function

$$\phi_{w_\varphi}(\mathbf{t})_i = w_\varphi \cdot \frac{e^{\mathbf{t}_i}}{\sum_{j=1}^{q} e^{\mathbf{t}_j}}.$$

The boost function ϕ_{w_φ} employs a clause weight w_φ as a learnable parameter so that the updates for the groundings are proportional to w_φ. Therefore, w_φ determines the magnitude of the update and thus reflects the impact of a clause. The changes to atoms that do not appear in a clause are set to zero. The boost function is applied row-wise to $\mathbf{M}$ as illustrated in the following example.

Example 2.7. *Given the clause* $\varphi_{AI} : \forall xy \neg \mathrm{AI(x)} \vee \neg \mathrm{Cit(x,y)} \vee \mathrm{AI(y)}$ *with the clause weight* w_{AI}*, the changes for this clause are* $\delta\mathbf{M}_{\varphi_{AI}} =$

$$w_{\mathrm{AI}} \cdot \begin{bmatrix} \delta_{\neg\mathrm{AI^x(a)}} & 0 & \dots & \delta_{\mathrm{AI^y(c)}} & 0 & \dots & \delta_{\neg\mathrm{Cit(a,c)}} \\ \delta_{\neg\mathrm{AI^x(a)}} & 0 & \dots & \delta_{\mathrm{AI^y(e)}} & 0 & \dots & \delta_{\neg\mathrm{Cit(e,a)}} \\ \delta_{\neg\mathrm{AI^x(a)}} & 0 & \dots & \delta_{\mathrm{AI^y(d)}} & 0 & \dots & \delta_{\neg\mathrm{Cit(c,d)}} \\ \vdots & & & \vdots & \vdots & & \\ \delta_{\neg\mathrm{AI^x(e)}} & 0 & \dots & \delta_{\mathrm{AI^y(f)}} & 0 & \dots & \delta_{\neg\mathrm{Cit(e,f)}} \end{bmatrix}$$

The values of $\delta\mathbf{M}_{\varphi_{AI}}$ *are calculated by* $\phi_{w_{\mathrm{AI}}}$*, for example:*

$$\delta_{\neg\mathrm{AI^x(a)}} = \phi_{w_{\mathrm{AI}}}(\mathbf{z})_a = -\frac{e^{-\mathbf{z}_{AI(a)}}}{e^{-\mathbf{z}_{AI(a)}} + e^{-\mathbf{z}_{Cit(a,c)}} + e^{\mathbf{z}_{AI(c)}}}$$

A clause enhancer is instantiated for each clause $\varphi \in \mathcal{K}$. Each clause enhancer computes updates $\delta\mathbf{M}_\varphi$ for a clause independently. The updates of all clause enhancers are finally added, resulting in a matrix $\delta\mathbf{M} = \sum_{\varphi\in\mathcal{K}} \delta\mathbf{M}_\varphi$. To apply the updates to the initial predictions, $\delta\mathbf{M}$ has to be added to $\mathbf{Y}$. The updates in $\delta\mathbf{M}$ can not directly be applied to the predictions $\mathbf{Y}$ of the GNN. Since the unary groundings $\mathbf{U}$ were joined with $\mathbf{B}$, multiple changes may be proposed for the same grounded unary atom. For example, for the grounded atom $\mathrm{AI(c)}$ the changes $\delta_{\neg\mathrm{AI^y(c)}}$ and $\delta_{\neg\mathrm{AI^x(c)}}$ are proposed, since c appears in first place of edge (a,c) and in second place of edge (c,e). Therefore, all updates for the same grounded atom are summed, reducing the size of $\mathbf{M}$ to the size of $\mathbf{U}$. To ensure that the updated predictions remain truth values in the range of $[0,1]$, the knowledge enhancer works with the preactivations $\mathbf{Z}$ of the GNN and applies the activation function σ to the updated preactivations $\mathbf{Z}'$ to obtain the final predictions: $\mathbf{Y}' = \sigma(\mathbf{Z}')$. Therefore, the knowledge enhancer transforms $\mathbf{Z}$ to $\mathbf{Z}'$ (with $\mathbf{Z},\mathbf{Z}' \in \mathbb{R}^{n\times m}$). Regarding the binary groundings, the values in $\mathbf{B}$ are set to a high positive value that results in one when σ is applied. In the last step, the updates by the knowledge enhancer are added to the preactivations $\mathbf{Z}$ of the GNN and passed to σ to obtain the updated predictions

$$\mathbf{Y}' = \sigma\left(\mathbf{Z}x + \sum_{\varphi\in\mathcal{K}} \delta\mathbf{U}_\varphi\right)$$

where $\delta\mathbf{U}_\varphi$ is the matrix obtained by extracting the changes to the unary predicates from $\delta\mathbf{M}_\varphi$.

3 Related Work

The field of knowledge graph completion is addressed from several research directions. Symbolic methods exist that conduct link prediction given a set of prior knowledge [Dou *et al.*, 2015] [Meilicke *et al.*, 2019]. Embedding-based methods [Dai *et al.*, 2020] are mostly sub-symbolic methods to obtain node embeddings that are used for knowledge graph completion tasks. Usually, their common objective is to find similar embeddings for nodes that are located closely in the graph. The majority of these methods only encodes the graph structure, but does not consider node-specific feature information [Abboud and Ceylan, 2021]. However, KeGNN is based on GNNs that are suited for learning representations of graphs attributed with node features. It stacks additional layers that interpret the outputs of the GNN in fuzzy logic and modify them to increase the satisfiability. Therefore, it is considered a neuro-symbolic method. In the multifaceted neuro-symbolic field, KeGNN can be placed in the category of knowledge-guided learning [Daniele and Serafini, 2020], where the focus lies on learning in the presence of additional supervision introduced as prior knowledge. Within this category, KeGNN belongs to the model-based approaches, where prior knowledge in the form of knowledge enhancement layers is an integral part of the model. Beyond, loss-based methods such as logic tensor networks [Badreddine *et al.*, 2022] exist that encode the satisfiability of prior knowledge as an optimization objective.

Further, in [DeLong *et al.*, 2023] neuro-symbolic approaches dealing with graph structures are classified into three categories. First, logically informed embedding approaches [Li *et al.*, 2023] [Jain *et al.*, 2021] use predefined logical rules that provide knowledge to a neural system, while both components are mostly distinct. Second, approaches for knowledge graph embedding with logical constraints [Fatemi *et al.*, 2019] [Guo *et al.*, 2016] use prior knowledge as constraints on the neural knowledge graph embedding method in order to modify predictions or embeddings. Thirdly, neuro-symbolic methods are used for learning rules for graph reasoning tasks [Hu *et al.*, 2020] [Qu *et al.*, 2021]. This allows for rule generation or confidence scores for prior knowledge and makes the models robust to exceptions or soft knowledge. KeGNN best falls into the second category, since the prior knowledge is interpreted in fuzzy logic to be integrated with the neural model and update the GNN's predictions. The idea of confidence values in category three shares the common property of relativating knowledge as with KeGNN's clause weights. However, even though KeGNN's clause weights introduce a notion of impact of a clause when predictions are made, they cannot directly be interpreted as the confidence in a rule. In the well-known *Kautz Taxonomy* [Kautz, 2022] that classifies neuro-symbolic approaches according to the integration of neural and symbolic modules, KeGNN falls best into the category `Neuro[Symbolic]` (Type 6) of fully-integrated neuro-symbolic systems that embed symbolic reasoning in a neural architecture.

4 Experimental Evaluation

To evaluate the performance of KeGNN, we apply it to the datasets Citeseer, Cora, PubMed and Flickr that are common benchmarks for node classification in a transductive setting. In the following, KeGNN is called KeGCN and KeGAT when instantiated to a GCN or a GAT, respectively. As additional baseline, we consider KeMLP, that stacks knowledge enhancement layers onto an MLP, as proposed in [Daniele and Serafini, 2022]. Further, the standalone neural models MLP, GCN and GAT are used as baselines. While Citeseer, Cora and PubMed are citation graphs that encode citations between scientific papers (as in Example 2.2), Flickr contains images and shared properties between them. All datasets can be modelled as homogeneous, labelled and attributed graphs as defined in Section 2.1. The set of prior logic for the knowledge enhancement layers is given explicitly. In this work, we encode the assumption that the existence of an edge for a node pair points to their membership to the same class and hence provides added value to the node classification task. In the context of citation graphs, this implies that two documents that cite each other refer to the same topic, while for Flickr, linked images share the same properties. Following this pattern for all datasets, a clause φ: $\forall xy : \neg \mathrm{Cls_i}(x) \vee \neg \mathrm{Link}(x, y) \vee \mathrm{Cls_i}(y)$ is instantiated for each node class $\mathrm{Cls_i}$. More details on the experiments are given in Appendix A. The implementation and the experiments is publicly available on Gitlab[3].

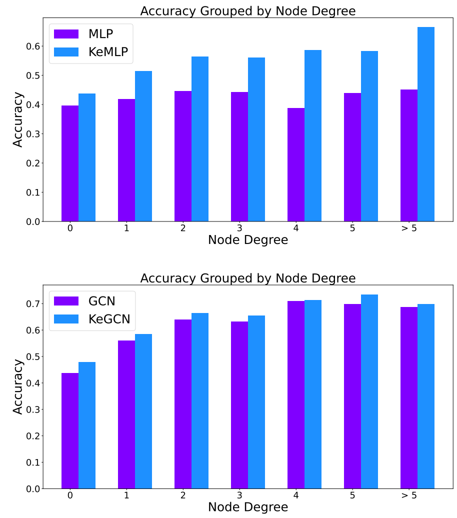

Figure (3) The accuracy grouped by the node degree for MLP vs. KeMLP (above) and GCN and KeGCN (below) on Citeseer.

4.1 Results

To compare the performance of all models, we examine the average test accuracy over 50 runs (10 for Flickr) for the knowledge enhanced models KeMLP, KeGCN, KeGAT and the standalone base models MLP, GCN, GAT on the named datasets. The results are given in Tab. 1 and visualized in Fig. 7 (see Appendix A.3). For Cora and Citeseer, KeMLP leads to a significant improvement over MLP (p-value of one-sided t-test $\ll 0.05$). In contrast, no significant advantage of KeGCN or KeGAT in comparison to the standalone base model is observed. Nevertheless, all GNN-based models are significantly superior to KeMLP for Cora. This includes not only KeGCN and KeGAT, but also the GNN baselines. For Citeseer, KeGAT and GAT both outperform KeMLP. In the case of PubMed, only a significant improvement of KeMLP over MLP can be observed, while the GNN-based models and their enhanced versions do not provide any positive effect. For Flickr, no significant improvement between the base model and the respective knowledge enhanced model can be observed. Nevertheless, all GNN-based models outperform KeMLP, reporting significantly higher mean test accuracies for KeGAT, GAT, GCN and KeGCN.

Exploitation of the Graph Structure

It turns out that the performance gap between MLP and KeMLP is larger than for KeGCN in comparison to the standalone GCN (or KeGAT vs. GAT, respectively). To explain this observation, we examine how the graph structure affects the prediction performance. Therefore, in Fig. 3 we analyze the accuracy grouped by the node degree for the entire graph

[3]https://gitlab.inria.fr/tyrex/kegnn

	MLP	KeMLP	GCN	KeGCN	GAT	KeGAT
Cora	0.7098 (0.0080)	0.8072 (0.0193)	0.8538 (0.0057)	**0.8587** (0.0057)	0.8517 (0.0068)	0.8498 (0.0066)
CiteSeer	0.7278 (0.0081)	0.7529 (0.0067)	0.748 (0.0102)	0.7506 (0.0096)	0.7718 (0.0072)	**0.7734** (0.0073)
PubMed	0.8844 (0.0057)	**0.8931** (0.0048)	0.8855 (0.0062)	0.8840 (0.0087)	0.8769 (0.0040)	0.8686 (0.0081)
Flickr	0.4656 (0.0018)	0.4659 (0.0012)	**0.5007** (0.0063)	0.4974 (0.0180)	0.4970 (0.0124)	0.4920 (0.0189)

Table (1) Average test accuracy of 50 runs (10 for Flickr). The standard deviations are reported in brackets.

for MLP vs. KeMLP and GCN vs. KeGCN [4]. It is observed that KeMLP performs better compared to MLP as the node degree increases. By contrast, when comparing GCN and KeGCN, for both models, the accuracy increases for nodes with a higher degree. This shows that rich graph structure is helpful for the node classification in general. Indeed, the MLP is a simple model that misses information on the graph structure and thus benefits from graph structure in the form of binary predicates contributed by KeMLP. On the contrary, standalone GNNs can process graph structure by using message passing techniques to transmit learned node representations between neighbors. The prior knowledge introduced in the knowledge enhancer is simple. It encodes that two neighbors are likely to be of the same class. An explanation for the small difference in performance is that GNNs may be able to capture and propagate this simple knowledge across neighbors implicitly, using its message passing technique. In other words we observe that, in this particular case, the introduced knowledge happens to be redundant for GNNs. However, the introduced knowledge significantly improves the accuracy of MLPs. In this context, we discuss perspectives for future work in Section 5.

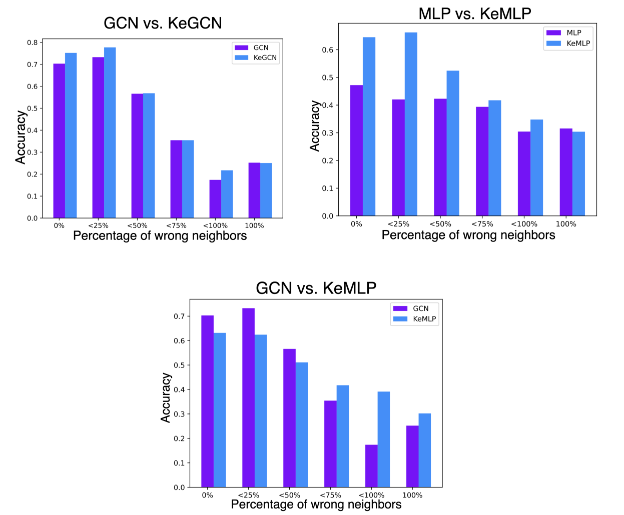

Figure (4) The accuracy grouped by the ratio of misleading first-order neighbors for GCN vs. KeGCN (left), MLP vs. KeMLP (right), GCN vs. KeMLP (below) on Citeseer.

Robustness to wrong knowledge

Furthermore, a question of interest is how the knowledge enhanced model finds a balance between knowledge and graph data in case of knowledge that is not consistent with the graph data. In other words, can the KeGNN successfully deal with nodes having mainly neighbors that belong to a different ground truth class and thus contribute misleading information to the node classification? To analyze this question, we categorize the accuracy by the proportion of misleading nodes in the neighborhood, see Fig. 4. Misleading nodes are nodes that have a different ground truth class than the node to be classified. It turns out that KeMLP is particularly helpful over MLP when the neighborhood provides the right information. However, if the neighborhood is misleading (if most or even all of the neighbors belong to a different class), an MLP that ignores the graph structure can lead to even better results. When comparing KeGCN and GCN, there is no clear difference. This is expected, since both models are equally affected by misleading nodes as they utilise the graph structure. Just as a GCN, the KeGCN is not necessarily robust to wrong prior knowledge since the GCN component uses the entire neighborhood, including the misleading nodes. When comparing GCN to KeMLP, see plot below in Fig.4, KeMLP is more robust to misleading neighbors. While GCN takes the graph structure as given and includes all neighbors equally in the embeddings by graph convolution, the clause weights in the knowledge enhancement methods provide a way to devalue knowledge. If the data frequently contradicts a clause, the model has the capacity to reduce the respective clause weight in the learning process and reduce its impact.

Clause Weight Learning

The clause weights learned during training provide insights on the updates made by a clause. The *clause compliance* (see

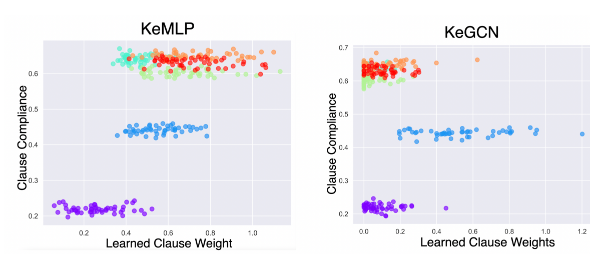

Figure (5) Learned clause weights vs. clause compliance for KeMLP (left) and KeGCN (right) on Citeseer.

[4]The findings for KeGAT are in line with those for KeGCN, see Fig. 8 in Section A.3

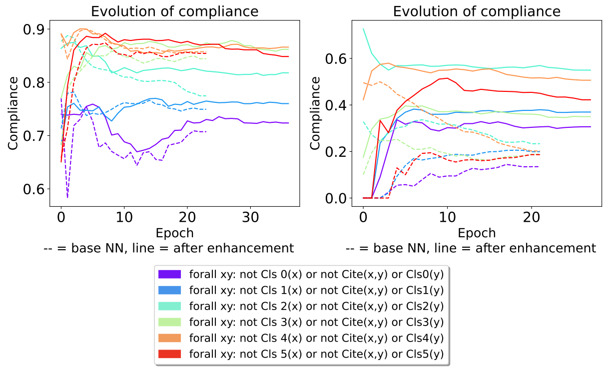

Figure (6) Clause compliance during training for GCN vs. KeGCN (left) and MLP vs. KeMLP (right) on Citeseer.

Appendix B) [Daniele and Serafini, 2020] measures how well the prior knowledge is satisfied in a graph. It can be calculated on the ground truth classes or the predicted classes. As a reference, we measure the clause compliance based on the ground truth labels in the training set. Fig. 5 displays the learned clause weights for KeGCN and KeMLP versus the clause compliance. For KeMLP, a positive correlation between the learned clause weights and the clause compliance on the training set is observed. This indicates that higher clause weights are learned for clauses that are satisfied in the training set. Consequently, these clauses have a higher impact on the updates of the predictions. In addition, the clause weights corresponding to clauses with low compliance values make smaller updates to the initial predictions. Accordingly, clauses that are rarely satisfied learn lower clause weights during the training process. In the case of KeGCN, the clause weights are predominantly set to values close to zero. This is in accordance with the absence of a significant performance gap between GCN and KeGCN. Since the GCN itself already leads to valid classifications, smaller updates are required by the clause enhancers.

Furthermore, we analyse how the compliance evolves during training to investigate whether the models learn predictions that increase the satisfaction of the prior knowledge. Fig. 6 plots the evolution of the clause compliance for the six clauses for GCN vs. KeGCN and MLP vs. KeMLP. It is observed that GCN and KeGCN yield similar results as the evolution of the compliance during training for both models is mostly aligned. For MLP vs. KeMLP the clause compliance of the prediction of the MLP converges to lower values for all classes than the clause compliance obtained with the KeMLP. This gives evidence that the knowledge enhancement layer actually improves the satisfiability of the prior knowledge. As already observed, the GCN is also able to implicitly satisfy the prior knowledge even though it is not explicitly defined.

5 Limitations and Perspectives

The method of KeGNN is limited in some aspects, which we present in this section. In this work, we focused on homogeneous graphs. In reality, however, graphs are often heterogeneous with multiple node and edge types [Yang *et al.*, 2022]. Adaptations are necessary on both the neural and the symbolic side to apply KeGNN to heterogeneous graphs. The restriction to homogeneous graphs also limits the scope of formulating complex prior knowledge. Eventually, the datasets used in this work and the set of prior knowledge are too simple for KeGNN to exploit its potential and lead to a significant improvement over the GNN. Experimental results show that knowledge encoded by the symbolic component leads to significant improvement over a model that is not capable to capture and learn that knowledge. This indicates that for more complex knowledge that is harder for a GNN to learn, KeGNN has the potential to bring higher improvements. A perspective for further work is the extension of KeGNN to more generic data structures such as incomplete and heterogeneous knowledge graphs in conjunction with more complex prior knowledge.

Another limitation of KeGNN is scalability. With an increasing number of stacked knowledge enhancement layers, the affected node neighborhood grows exponentially, which can lead to significant memory overhead. This problem is referred as neighborhood explosion [Duan *et al.*, 2022] and is particularly problematic in the context of training on memory-constrained GPUs. This affects both the GNN and the knowledge enhancement layers that encode binary knowledge. Methods from scalable graph learning [Fey *et al.*, 2021] [Zeng *et al.*, 2020] [Hamilton *et al.*, 2017] represent potential solutions for the neighborhood explosion problem in KeGNN.

Furthermore, limitations appear in the context of link prediction with KeGNN. For link prediction, a neural component is required that predicts fuzzy truth values for binary predicates. At present, KeGNN can handle clauses containing binary predicates, but their truth values are initialized with artificial predictions, where a high value encodes the presence of an edge. This limits the application of KeGNN to datasets for which the graph structure is complete and known a priori.

6 Conclusion

In this work, we introduced KeGNN, a neuro-symbolic model that integrates GNNs with symbolic knowledge enhancement layers to create a fully differentiable end-to-end model. This allows the use of prior knowledge to improve node classification while exploiting the expressive representations of a GNN. Experimental studies show that the inclusion of prior knowledge has the potential to improve simple neural models (as observed in the case of MLP). However, the knowledge enhancement of GNNs is harder to achieve on the underlying and limited benchmarks for which the injection of simple knowledge concerning local neighborhood is redundant with the representations that GNNs are able to learn. Nevertheless, KeGNN has not only the potential to improve graph completion tasks from a performance perspective, but also to increase interpretability through clause weights. This work is a step towards a holistic neuro-symbolic method on incomplete and noisy semantic data, such as knowledge graphs.

A Experiment Details

A.1 Implementation

The implementation of KeGNN and the described experiments is publicly available on GitLab[5]. The code is based on PyTorch [Paszke *et al.*, 2019] and the graph learning library PyTorch Geometric [Fey and Lenssen, 2019]. The Weights & Biases tracking tool [Biewald, 2020] is used to monitor the experiments. All experiments are conducted on a machine running an Ubuntu 20.4 equipped with an Intel(R) Xeon(R) Silver 4114 CPU 2.20GHz processor, 192G of RAM and one GPU Nvidia Quadro P5000.

A.2 Datasets

Tab. 2 gives an overview of the named datasets in this work. The datasets are publicly available on the dataset collection[6] of PyTorch Geometric [Fey and Lenssen, 2019]. For the split into train, valid and test set, we take the predefined splits in [Chen *et al.*, 2018] for the citation graphs and in [Zeng *et al.*, 2020] for Flickr. Word2Vec vectors [Adewumi *et al.*, 2020] are used as node features for the citation graphs and image data for Flickr. Fig. 1 visualizes the graph structure of the underlying datasets in this work as a homogeneous, attributed and labelled graph (on the example of Citeseer).

A.3 Results

The average test accuracies obtained for the node classification experiments on Cora, Citeseer, PubMed and Flickr over all tested models are visualized in Fig. 7. The average runtimes per epoch on the Citeseer dataset are compared for all models in Tab 3. The runtimes were calculated for models with three hidden layers and three knowledge enhancement layers in full-batch training. It can be noted that the knowledge enhancement layers lead to increased epoch times since the model complexity is higher.

Model	Avg Epoch Time
MLP	0.02684
GCN	0.03109
GAT	0.06228
KeMLP	0.04304
KeGCN	0.03747
KeGAT	0.08384

Table (3) Comparison of the average epoch times for all models on the Citeseer dataset.

Fig. 8 shows the accuracy grouped by node degree for GAT vs. KeGAT.

[5]https://gitlab.inria.fr/tyrex/kegnn

[6]https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

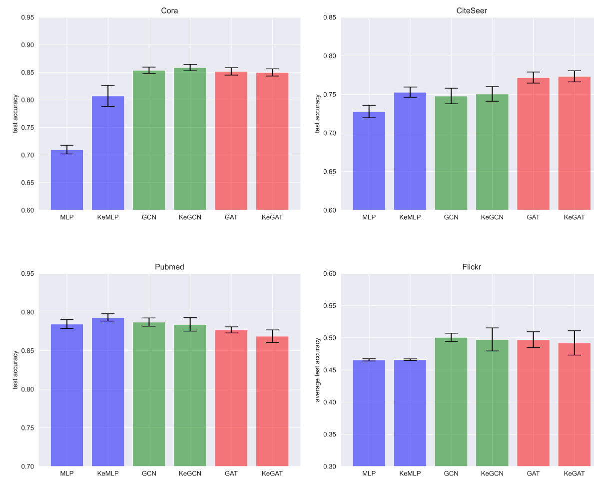

Figure (7) Average test accuracies over 50 runs for Cora, CiteSeer and PubMed and 10 runs on Flickr. Error bars denote standard deviation.

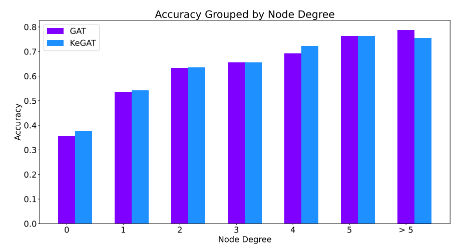

Figure (8) The accuracy grouped by the node degree for GAT vs. KeGAT on Citeseer.

A.4 Hyperparameter Tuning

KeGNN contains a set of hyperparameters. Batch normalization [Ioffe and Szegedy, 2015] is applied after each hidden layer of the GNN. The Adam optimizer [Kingma and Ba, 2015] is used as optimizer for all models. Concerning the hyperparameters specific to the knowledge enhancement layers, the initialization of the preactivations of the binary predicates (which are assumed to be known) is taken as a hyperparameter. They are set to a high positive value for edges that are known to exist and correspond to the grounding of the binary predicate. Furthermore, different initializations of clause weights and constraints on them are tested. Moreover, the number of stacked knowledge enhancement layers is a hyperparameter. We further allow the model to randomly neglect a proportion of edges by setting an edges drop rate parameter. Further, we test whether the normalization of the edges with the diagonal matrix $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{i,j}$ (with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$) is helpful.

To find a suitable set hyperparameters for each dataset and model, we perform a random search with up to 800 runs and 48h time limit and choose the parameter combination which leads to the highest accuracy on the validation set. The hyperparameter tuning is executed in Weights and Biases [Biewald, 2020]. The following hyperparameter values are tested:

- Adam optimizer parameters: β_1: 0.9, β_2: 0.99, ϵ: 1e-07

Name	#nodes	#edges	#features	#Classes	train/valid/test split
Citeseer	3,327	9,104	3,703	6	1817/500/1000
Cora	2,708	10,556	1,433	7	1208/500/1000
PubMed	19,717	88,648	500	3	18217/500/1000
Flickr	89,250	899,756	500	7	44624/22312/22312

Table (2) Overview of the datasets Citeseer, Cora, PubMed and Flickr

- Attention heads: $\{1, 2, 3, 4, 6, 8, 10\}$
- Batch size: $\{128, 512, 1024, 2048,$ full batch$\}$
- Binary preactivation: $\{0.5, 1.0, 10.0, 100.0, 500.0\}$
- Clause weights initialization: $\{0.001, 0.1, 0.25, 0.5,$ random uniform distribution on [0,1)$\}$
- Dropout rate: 0.5
- Edges drop rate: random uniform distribution $[0.0, 0.9]$
- Edge normalization: $\{$true, false$\}$
- Early stopping: δ_{min} : 0.001, patience: $\{$1, 10, 100$\}$
- Hidden layer dimension: $\{$32, 64, 128, 256$\}$
- Learning rate: random uniform distribution $[0.0001, 0.1]$
- Clause weight clipping: w_{min} : 0.0, w_{max}: random uniform distribution: $[0.8, 500.0]$
- Number of knowledge enhancement layers: $\{1, 2, 3, 4, 5, 6\}$
- Number of hidden layers: $\{2, 3, 4, 5, 6\}$

The obtained parameter combinations for the models KeMLP, KeGCN and KeGAT for Cora, Citeseer, PubMed and Flickr are displayed in Tab. 5 and Tab. 4. The reference models MLP, GCN and GAT are trained with the same parameter set as the respective knowledge enhanced models.

B Clause Weight Evaluation

The *clause compliance* [Daniele and Serafini, 2020] indicates the level of satisfaction of a clause in the data in this experimental setting. Given a clause φ, a class Cls_m, the set of training nodes $\mathbf{V}_{\text{train}}$, the set of nodes of the class $\mathrm{Cls}_{\mathrm{m}}$: $\mathbf{V}_m = \{v_i | v_i \in \mathbf{V}_{\text{train}} \wedge \mathrm{Cls}(v_i) == \mathrm{m}\}$, and the neighborhood $\mathcal{N}(v_i)$ of v_i, the clause compliance on graph $\mathbf{G}$ is defined as follows:

$$\mathrm{Compliance}(\mathbf{G}, \varphi) = \frac{\sum_{v_i \in \mathbf{V}_k} \sum_{v_j \in \mathcal{N}(v)} \mathbf{1}[\text{ if } v_j \in \mathbf{V}_m]}{\sum_{v_i \in \mathbf{V}_m} |\mathcal{N}(v_i)|} \tag{1}$$

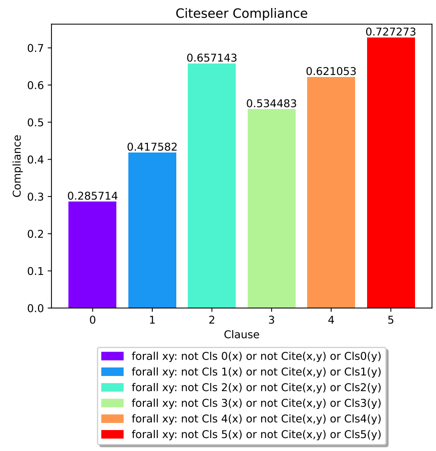

Figure (9) The clause compliance on the ground truth graph on the training set for Citeseer.

In other words, the clause compliance counts how often among nodes of a class the neighboring nodes are of the same class. [Daniele and Serafini, 2020]

	PubMed			Flickr		
Parameter	**KeMLP**	**KeGCN**	**KeGAT**	**KeMLP**	**KeGCN**	**KeGAT**
adam beta 1	0.9	0.9	0.9	0.9	0.9	0.9
ada beta 2	0.99	0.99	0.99	0.99	0.99	0.99
adam epsilon	1e-07	1e-07	1e-07	1e-07	1e-07	1e-07
attention heads	-	-	8	-	-	8
batch size	1024	full batch	1024	128	1024	2048
binary preactivation	10.0	1.0	10.0	10.0	500.0	500.0
clause weight initialization	0.001	random	0.5	0.001	0.001	0.1
dropout rate	0.5	0.5	0.5	0.5	0.5	0.5
edges drop rate	0.22	0.66	0.07	0.2	0.24	0.12
epochs	200	200	200	200	200	200
early stopping enabled	true	true	true	true	true	true
early stopping min delta	0.001	0.001	0.001	0.001	0.001	0.001
early stopping patience	100	10	10	10	10	100
hidden channels	256	256	256	32	128	64
learning rate	0.057	0.043	0.016	0.001	0.016	0.0039
max clause weight	350.0	322.0	118.0	55.0	135	113.0
min clause weight	0.0	0.0	0.0	0.0	0	0.0
normalize edges	false	false	true	true	true	false
KE layers	2	1	5	1	4	1
hidden layers	4	2	2	2	4	3
runs	50	50	50	10	10	10
seed	1234	1234	1234	1234	1234	1234

Table (4) Hyperparameters and experiment configuration for PubMed and Flickr

	Cora			CiteSeer		
Parameter	**KeMLP**	**KeGCN**	**KeGAT**	**KeMLP**	**KeGCN**	**KeGAT**
adam beta 1	0.9	0.9	0.9	0.9	0.9	0.9
ada beta 2	0.99	0.99	0.99	0.99	0.99	0.99
adam epsilon	1e-07	1e-07	1e-07	1e-07	1e-07	1e-07
attention heads	-	-	1	-	-	3
batch size	512	512	full batch	128	full batch	1024
binary preactivation	10.0	500.0	1.0	10.0	0.5	0.5
clause weight initialization	0.5	random	0.5	0.5	0.25	0.1
dropout rate	0.5	0.5	0.5	0.5	0.5	0.5
edges drop rate	0.47	0.17	0.27	0.01	0.35	0.88
epochs	200	200	200	200	200	200
early stopping enabled	true	true	true	true	true	true
early stopping min delta	0.001	0.001	0.001	0.001	0.001	0.001
early stopping patience	1	1	10	10	10	10
hidden channels	32	256	64	256	128	32
learning rate	0.026	0.032	0.033	0.028	0.037	0.006
max clause weight	104.0	254.0	250.0	34.0	243.0	110.0
min clause weight	0.0	0.0	0.0	0.0	0.0	0.0
normalize edges	true	false	true	true	false	true
KE layers	4	2	1	1	3	2
Hidden layers	2	2	2	2	5	2
runs	50	50	50	50	50	50
seed	1234	1234	1234	1234	1234	1234

Table (5) Hyperparameter and experiment configuration for Citeseer and Cora

References

[Abboud and Ceylan, 2021] Ralph Abboud and Ismail Ilkan Ceylan. Node classification meets link prediction on knowledge graphs. https://arxiv.org/abs/2106.07297, 2021.

[Adewumi *et al.*, 2020] Tosin P. Adewumi, Foteini Liwicki, and Marcus Liwicki. Word2vec: Optimal hyper-parameters and their impact on nlp downstream tasks. https://arxiv.org/abs/2003.11645, 2020.

[Badreddine *et al.*, 2022] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, feb 2022.

[Biewald, 2020] Lukas Biewald. Experiment tracking with weights and biases. https://www.wandb.com/, 2020. Software available from wandb.com.

[Chen *et al.*, 2018] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR (Poster)*. OpenReview.net, 2018.

[Dai *et al.*, 2020] Yuanfei Dai, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5), 2020.

[Daniele and Serafini, 2020] Alessandro Daniele and Luciano Serafini. Neural networks enhancement with logical knowledge. https://arxiv.org/abs/2009.06087, 2020.

[Daniele and Serafini, 2022] Alessandro Daniele and Luciano Serafini. Knowledge enhanced neural networks for relational domains. https://arxiv.org/abs/2205.15762, 2022.

[DeLong *et al.*, 2023] Lauren Nicole DeLong, Ramon Fernández Mir, Matthew Whyte, Zonglin Ji, and Jacques D. Fleuriot. Neurosymbolic ai for reasoning on graph structures: A survey. https://arxiv.org/abs/2302.07200, 2023.

[Dou *et al.*, 2015] Dejing Dou, Hao Wang, and Haishan Liu. Semantic data mining: A survey of ontology-based approaches. In *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*, pages 244–251, 2015.

[Duan *et al.*, 2022] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[Fatemi *et al.*, 2019] Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:3526–3533, 07 2019.

[Fey and Lenssen, 2019] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[Fey *et al.*, 2021] Matthias Fey, Jan E. Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3294–3304. PMLR, 18–24 Jul 2021.

[Guo *et al.*, 2016] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 192–202, Austin, Texas, November 2016. Association for Computational Linguistics.

[Hamilton *et al.*, 2017] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[Hu *et al.*, 2020] Yuwei Hu, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Li, Zheng Zhang, Zhiru Zhang, and Yida Wang. Featgraph: A flexible and efficient backend for graph neural network systems. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2020.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[Jain *et al.*, 2021] Nitisha Jain, Trung-Kien Tran, Mohamed H. Gad-Elrab, and Daria Stepanova. Improving knowledge graph embeddings with ontological reasoning. In *The Semantic Web – ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings*, page 410–426, Berlin, Heidelberg, 2021. Springer-Verlag.

[Kautz, 2022] Henry A. Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. https://onlinelibrary.wiley.com/doi/10.1002/aaai.12036, 2022.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

[Klement *et al.*, 2013] E.P. Klement, R. Mesiar, and E. Pap. *Triangular Norms*. Trends in Logic. Springer Netherlands, 2013.

[Li *et al.*, 2023] Weidong Li, Rong Peng, and Zhi Li. Knowledge graph completion by jointly learning structural features and soft logical rules. *IEEE Transactions on Knowledge and Data Engineering*, 35(3):2724–2735, 2023.

[Liu *et al.*, 2021] Weiwen Liu, Yin Zhang, Jianling Wang, Yun He, James Caverlee, Patrick Chan, Daniel Yeung, and Pheng-Ann Heng. Item relationship graph neural networks for e-commerce. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–15, 03 2021.

[Ma and Tang, 2021] Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021.

[Meilicke *et al.*, 2019] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 3137–3143. AAAI Press, 2019.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[Qu *et al.*, 2021] Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. {RNNL}ogic: Learning logic rules for reasoning on knowledge graphs. In *International Conference on Learning Representations*, 2021.

[Sanchez-Gonzalez *et al.*, 2018] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4470–4479. PMLR, 10–15 Jul 2018.

[Susskind *et al.*, 2021] Zachary Susskind, Bryce Arden, Lizy K. John, Patrick Stockton, and Eugene B. John. Neuro-symbolic AI: an emerging class of AI workloads and their characterization. *CoRR*, abs/2109.06133, 2021.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[Wu *et al.*, 2020] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1054–1061, Apr. 2020.

[Wu *et al.*, 2021] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[Wu *et al.*, 2022] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022.

[Yang *et al.*, 2016] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 40–48. JMLR.org, 2016.

[Yang *et al.*, 2022] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. Simple and efficient heterogeneous graph neural network. https://arxiv.org/abs/2207.02547, 2022.

[Zadeh, 1988] L.A. Zadeh. Fuzzy logic. *Computer*, 21(4):83–93, 1988.

[Zeng *et al.*, 2020] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.