Scaling Relational Graph Convolutional Network Training with Graph Summaries and Entity Embedding Transfer

Tiddo Loos[2574974]
t.j.loos@student.vu.nl

Vrije Universiteit, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

Abstract. Relational Graph Convolutional Network (R-GCN) training on real-world graphs is challenging. Storing gradient information during R-GCN training on real-world graphs, exceeds available memory on most single devices. Recent work demonstrated to scale R-GCN training with a summary graph. The appropriate graph summarization technique is often unknown and graph and task dependent. Overcoming this problem, we propose R-GCN pre-training on multiple graph summaries, produced with attribute and (k)-forward bisimulation summarization techniques. With pre-training on graph summaries, multiple entity embeddings and one set R-GCN weights can be obtained. We applied *Summation*, *Multi-Layer Perceptron* and *Multi-Head Attention* models to transfer multiple entity embeddings and R-GCN weights to a new R-GCN model. With the new R-GCN model we conducted full-graph training for entity type prediction. Our contribution to existing research is three-fold, as this work demonstrated how: graph summaries reduce parameters for R-GCN training, while maintaining or improving R-GCN performance; the creation of graph summaries can be included in R-GCN training to maintain or improve R-GCN performance, while reducing computational time; graph summaries in combination with *Multi-Layer Perceptron* and *Multi-Head Attention* can be applied to scale R-GCN training and maintain or improve R-GCN performance, while freezing the gradients of the R-GCN weights after summary graph pre-training. The code and datasets are available at GitHub.

Keywords: Knowledge Graph · Graph Summary · Scaling · Entity Embedding · Embedding Model

Table of Contents

1 Introduction

A (knowledge) graph is a relational data representation, expressing relations between entities. The Resource Description Framework (RDF) is a common framework to store relational data, such as web data [25]. A subject (entity), predicate (relation) and object (entity) are the building blocks of the RDF-triple [30]. A graph can be considered a data collection of RDF-triples. The RDF structure of the data enables a simple model of the relation between subject and object and information deduction with logical inference due to the semantics [1]. An entity/node[1] in the graph can have an entity type denoted by an *rdf:type*[2] relation. An example of a *subject-predicate-object* RDF-triple is: *Tarantino directed Kill Bill*. Looking at the RDF-triple example, entity types can be assigned as: *Tarentino rdf:type director, Kill Bill rdf:type movie*.

Building a knowledge graph, commonly, graph data is collected or added with the use of manually, semi-automated and automated methods [9, 15]. DBPedia, Wikidata and Yago are examples of constructed knowledge graphs and are impressive considering their size and collection efforts. However, the problem of incompleteness and missing data remains in these graphs. Missing data regarding graphs, comprise missing edges in the graph [41]. Fundamentals of Statistical Relational Learning, which is a specific field in graph learning, are link prediction and entity type prediction. Link prediction in graph learning equals to appending a *subject-predicate-object* RDF-triple. The aim of entity type prediction is to complement an entity with an *rdf:type* label. Entity type prediction for graphs nodes, is a transductive learning tasks, as the training and evaluation data both are encountered by the model. Type labels are pruned, while the graph nodes, for which the prediction is made, remain part of the training data [21].

Graphs, containing multiple entity and relation types, are called heterogeneous graphs. Significant research has been conducted on modeling heterogeneous graphs for relation type and entity type prediction. On these tasks, a well-performing model is the Relational Graph Convolutional Network[31] (R-GCN). The R-GCN model creates convolution kernels to learn relation-specific weights. Entity vector representations, called entity (or node) embeddings, can be constructed with R-GCN. R-GCN training on large graphs is computationally costly in terms of computational resource, as for every relation there exists a weight matrix. Another reason is that for every entity in the graph there exists an embedding. The entity embeddings are trainable parameters. Gradient information of the R-GCN weights and the entity embeddings need to be stored during training. Therefore, training the R-GCN model with real-world graphs, exceeds the available memory on most single devices. Following from the fact that R-GCN training on real-world graphs is computationally costly, R-GCN training time can be extensive as well.

[1] *entity* and *node* will be used interchangeably, as each entity in the RDF-triple is a graph node

[2] short for: `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

Considering computational resources, such as memory and time, scaling R-GCN training is a relevant challenge to facilitate R-GCN training with large graphs. This research builds on the concept of scaling R-GCN training with graph summaries, proposed in previous research[14]. Summary graphs are smaller graphs that express the structural characteristics of the original graph (where the summary graph originates from). In this research we use the attribute and (k)-forward bisimulation summarization techniques to create graph summaries. Previous research[14] on scaling R-GCN with graph summaries, fails to conclude which graph summarization technique is considered appropriate for summarizing heterogeneous graphs. As we do not know what graph summary is suitable for scaling R-GCN training for each data graph, we investigate models which exploit pre-training with multiple graph summaries.

From pre-training the R-GCN model with multiple summary graphs, we obtain multiple entity embeddings and one set of pre-trained R-GCN weights. After pre-training on summary graphs, the node embeddings and R-GCN weights are transferred to a new R-GCN model to conduct full-batch training on the original graph for the entity type prediction task. Multiple entity embeddings are transferred by combining these into one entity embedding fitting the original graph. The idea is that, by combining the entity embeddings, we enable the R-GCN model to exploit elements of different graph summaries. To transfer and combine entity embeddings of different graph summaries we propose the following strategies: *Summation*, *Multi-Layer Perceptron* and *Multi-Head Attention*. The *Summation* model transfers multiple entity embeddings by summing the entity embeddings, produced with summary graph pre-training, into one. The *Multi-Layer Perceptron* model transforms multiple summary graph entity embeddings with two matrix multiplications to produce one entity embedding. The *Multi-Head Attention* runs the multiple summary graph entity embeddings through multiple attention heads in parallel. The output of the attention heads are then concatenated and linearly transformed into an attended entity embedding.

With graph summaries we expect the R-GCN to learn entity embeddings that relate to the entity embeddings of the original graph nodes. By transferring the entity embeddings created with summary graph training, parameters may be reduced as the entity embedding for the original graph is constructed from smaller summary node entity embeddings. Previous research[14] measures that transferring R-GCN weights functions as jump start in the learning process. Therefore, by transferring the entity embeddings and R-GCN weights, we may scale R-GCN training, by reducing trainable parameters and training time.

Literature points out that the attribute summarization technique is in most cases a viable option for summarizing heterogeneous graphs. In terms of compressing heterogeneous graphs, the attribute summarization performs well [8]. Compression of the graph in graph summarization, is desired considering the size of real-world graphs and the goal of scaling R-GCN training. R-GCN training on a smaller (summary) graph becomes more manageable in terms of computational resources. However, heavily compressing a graph may result in a poor representation of the original graph structure.

1.1 Research Focus

The reported experiments in this work try to answer the research question: *How can multiple graph summaries scale R-GCN training for entity type prediction in graphs?* Furthermore, we try to answer the sub-questions:

- *How does a single graph summary influence scaling of R-GCN training and R-GCN performance on entity type prediction?*
- *How do pre-trained entity embeddings and R-GCN weights, obtained with summary graph training, influence scaling of R-GCN training and R-GCN performance on entity type prediction?*
- *How can R-GCN training be scaled and R-GCN performance, on entity type prediction, be maintained or improved, while freezing the gradient of R-GCN weights after pre-training on summary graphs?*

1.2 Contribution

Our contribution to existing research is three-fold, as this work demonstrated how: graph summaries reduce parameters for R-GCN training, while maintaining or improving R-GCN performance; the creation of graph summaries can be included in R-GCN training to maintain or improve R-GCN performance, while reducing computational time; graph summaries in combination with *Multi-Layer Perceptron* and *Multi-Head Attention* can be applied to scale R-GCN training and maintain or improve R-GCN performance, while freezing the gradients of the R-GCN weights after summary graph pre-training.

2 Related Work: Scaling Graph Learning

Relevant research on scaling graph training has been reported [16, 20, 23, 32, 35]. Facilitating full-batch (whole-graph) training on multiple devices in parallel, the NeuGraph[23] and Roc[16] frameworks were proposed. NeuGraph[23] and Roc[16] propose partitioning an input graph and apply full-batch training on the graph partitions across multiple compute nodes on a single server. In NeuGraph[23] and Roc[16], the graph model parameters are replicated on the compute nodes. The device nodes share feature information to complete each training iteration. Sharing feature information at each model layer is considered a drawback. The feature communication can be complex, as the communication depends on edge connections between nodes in different graph partitions [32].

Contrary to scaling graph training by applying full-batch training on graph partitions in parallel, scaling graph training by conducting mini-batch training on graph partitions in parallel has been researched [20, 35]. The goal of parallel mini-batch training, is to enable compute nodes on a single server to complete a mini-batch iteration without sharing features across compute nodes during the training iteration. A proposed architecture[35], demonstrates parallel training by creating a subset in the graph partition for each available compute node. For each graph partition, accessible neighboring nodes over *n-hops* are included, where n is set to be the number of layers in the graph model. Therefore, neighbor information over *n-hops* for each node in the graph partition is available, without having to share node information across devices. Through message passing, gradients are computed and with *AllRedcue* gradients are shared. The model is updated by averaging the gradients. The proposed method[35] speeds up training 16x on the link prediction task on the FB15k-237 and ogbl-citation2, while achieving comparable performance to related work. Accomplishing a 16x speed up, 8 computation nodes in parallel were used during training.

Another method for scaling graph learning, is PaGraph[20]. PaGraph reduces data loading time regarding node feature movement from CPU to GPU, during parallel mini-batch training. Through the use of a cache, PaGraph stores frequent accessed node information on the GPU's free space. Furthermore, the paper[20] discusses a graph partition algorithm to divide workload evenly across the devices. PaGraph achieves up to a 96.8% reduction of data loading time during each training epoch, while the Graph Neural Network and the Graph Convolution Network perform comparable to related work in terms of accuracy.

3 Graph Summarization

In this chapter we explain graph summarization. First, we provide a background on graph summarization techniques by discussing related work. Then, we define core concepts relevant to graph summarization techniques applied in this research. Using the core concepts, we formally define a Summarization Function with which graph summaries are created. After that, we explain the approximate graph summarization and attribute summaries relevant to this work. At last, we cover precise graph summarization and the (k)-forward bisimulation, a precise graph summarization technique.

Remark 1. In related research a resulting graph of graph summarization is often referred to as *'super graph'*. This research exploits graph summarization techniques that produce a *'super graph'*. However, we use *'summary graph'* to refer to the resulting product from graph summarization and *'original graph'* is used to refer to the graph where the *summary graph* originates from. Evidently, we use 'original nodes' and 'summary nodes' to refer to the set of nodes in the 'original graph' and 'summary graph' respectively.

3.1 Related Work: Graph Summarization Techniques

Graph summaries have benefits in various graph tasks like clustering [11], classification [17], and outlier detection [38]. Graph summarization could make tasks more manageable, as graph summarization reduces the volume of the graph and thus its memory footprint while preserving its structure. The graph summarization technique and its resulting summary graph vary in terms of compression and complexity. Characteristics of a graph summary depend on the summarization technique and on the original graph which can be either a weighted, directed, undirected and heterogeneous graph [22]. Literature proposes summarization methods that require different input. For example, some graph summarization methods require static plain graphs (unlabeled nodes end edges) [18, 36], some require static labeled graphs [4, 8, 36, 37]. Others, less commonly, handle dynamic graphs [33]. As the term dynamic implies, dynamic graphs change overtime. For example, when a new user is added to a social network, a new node for that person is added. When this person adds another person to its friends list a new edge is created, 'connecting' these two nodes.

As tasks differ, literature proposes different graph summarization techniques that build on specific characteristics of a graph summary. Widely researched graph summarization techniques are: *Grouping (attribute based)* [4, 8, 46], a technique that divides nodes into subset of nodes based on the structural characteristics or edge (attribute) types; *Bit Compression based* [33], where the goal is to describe the data of the original graph in as few bits as possible. Desired is that from the summary the original can be reconstructed lossless; *Simplification or sparsification based*, this technique banks on pruning less important nodes, to create a sparsified graph [18, 36]; and *influence based*, which tries to reveal the flow-based influence of nodes in large graphs on the rest of the nodes. As the

node influence is quantified, the summarization task becomes an optimization problem. The resulting summarization depends on most influential nodes [37].

Despite that it can be argued that every task requires a specific graph summary (and technique), the FLUID framework[4] is proposed. FLUID is a framework to create flexible graph summaries for data graphs. FLUID exploits commonalities in structural graph summaries and enables to quickly define, adapt and compare graph summaries for different purposes. Graph summaries are created with FLUID by exposing structural equivalent parts of a graph by applying bisimulation. Summarization techniques that use bisimulation for partitioning belong to the Precise Graph Summarization domain. The graph summary is precise if every path in the resulting summary graph exists in the original graph [8]. The FLUID framework offers a (k)-forward bisimulation summarization implementation, which takes neighboring node characteristics into account over k-hops (steps).

Yet another summarization technique is the Approximate Graph Summarization. A graph summary is approximate in the sense that nodes are portioned on a specific characteristic while ignoring others, making the partition an approximation for the graph. Contrary to the (k)-forward bisimulation, the approximate graph summaries only account for local (1-hop) node characteristics when creating the graph summary. Previous work[8] suggests that approximate graph summaries are to most viable option for summarizing heterogeneous graphs in most cases. Precise summary graphs depend on equivalent structures, while heterogeneous graphs express an inconsistent schema. Therefore, precise graph summaries could result in a very low compression rates until the point that there is no benefit of graph summarization.

3.2 Graph Summarization Concepts

In this research we focus on heterogeneous graphs where the graph contains multiple entity and relation types. We define a (knowledge) graph in Definition 1, which involves heterogeneous RDF-triple based graphs.

Definition 1. ***Graph*** *(adapted from [12])*
We define a Graph as a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$. $\mathcal{V}$ *is a set of nodes representing entities, and* $\mathcal{E}$ *a set of typed edges between the nodes. A function* $\tau : \mathcal{V} \rightarrow 2^{\mathcal{T}}$ *assigns one or more types to every node, where* $\mathcal{T}$ *is a set of entity types. Each edge in* $\mathcal{E}$ *corresponds to a relation between two nodes* v_i *and* $v_j \in \mathcal{V}$*, denoted by* $r(v_i, v_j)$*, where* $r \in \mathcal{R}$ *is a relation type.*

A core concept of graph summarization techniques that we exploit in this research, is creating subsets of graph nodes based on a similarity [4, 8]. Creating subsets from a larger set is referred to as creating a partition. A partition, in the current research, concerns subsets of graph nodes. We formally define a partition in Definition 2.

Definition 2. ***Partition***
Partition $\mathcal{P}$ of a set $\mathcal{X}$ is a division of elements of $\mathcal{X}$ into non-empty, disjoint subsets covering set $\mathcal{X}$. Therefore, $\mathcal{P}$ has the following properties:

- *Subsets are non-empty,* $\forall x_i \in \mathcal{P},\ x_i \neq \emptyset$
- *Subsets are disjoint,* $\forall x_i, x_j \in \mathcal{P},\ i \neq j \rightarrow x_i \wedge x_j = \emptyset$
- *Subsets exactly cover the original* $\mathbf{U}_{x_i \in \mathcal{P}}\, x_i = \mathcal{X}$

In order to create a partition, we formally define a subset mapping in Definition 3. With the subset mapping $\mathcal{S}$, a mapping from graph nodes to a subset in $\mathcal{Z}$ can be be specified. The subset mapping can account for requirements regarding the mapping from the graph node to a subset.

Definition 3. ***Subset Mapping*** *(adapted from [8])*
Let $G_o = (\mathcal{V}_o, \mathcal{E}_o, \mathcal{R}, \mathcal{T})$ be a graph (Definition 1). Let $\mathcal{Z}$ be a set of empty subsets. We define $\mathcal{S}$ to be a subset mapping that maps each node in $\mathcal{V}_o$ to a subset in $\mathcal{Z}$ such that:

$$\mathcal{S} \subseteq \mathcal{V}_o \times \mathcal{Z}$$

Graph Homomorphism Graph summarization techniques aim to abstract the graph and mirror its structure. Complexity of summarizing a graph increases if the graph has an irregular structure containing many relation and entity types. To ensure that the graph summary captures the structure of the original graph, the graph summary should act as a graph homomorphism [8]. A graph is homomorphic to another graph if there exists a mapping of nodes that matches edges from the first to the second graph. Mathematically, graph homomorphism can be considered a function that maps adjacent vertices of one graph to adjacent vertices of a second graph. Therefore, a map between the two vertex sets of two the graphs exists respecting each other's structure. An example of homomorphic graph summarization is given in Figure 1.

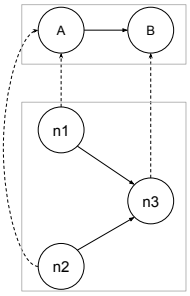

Figure 1. An original graph and its summary graph. The dotted lines indicate the mapping from the original nodes (n_1, n_2, n_3) to the summary nodes(A, B).

The dotted lines in Figure 1 represent the mapping from original node to summary node. The nodes *n1* and *n2* are represented in the upper summary graph by *A* and *n3* by *B*. Also, the paths from *n1* and *n2* to *n3* can be taken from *A* to *B*. As is the principle of graph homomorphism, in Figure 1 a mapping exists from summary nodes to original nodes that matches the edge structure of the original graph.

Summarization Function Considering a graph (Definition 1), the concept of graph homomorphism and the definition of a subset mapping (Definition 3), we define the Summarization Function (Definition 4). We formally define the Summarization Function, which takes any original graph G_o and any subset mapping $\mathcal{S}$ as input, to produce a tuple. The Summarization Function produces a summary graph G_s and a mapping M_s. According to subset mapping $\mathcal{S}$, the Summarization Function creates a partition $\mathcal{P}$ (Definition 2) by allocating the original nodes to its mapped subsets. We define the set of non-empty subsets in $\mathcal{Z}$, to be the partition $\mathcal{P}$. For each subset in $\mathcal{P}$ a summary node is created, resulting in summary node set $\mathcal{V}_s$. Each original node is now represented by a summary node, as each original node is in a subset and each subset is represented by a summary node. The graph summary is constructed by substituting original nodes with the representing summary nodes, respecting the edge connections of the original graph. As a partition $\mathcal{P}$ is created according to subset mapping $\mathcal{S}$, we define G_s to be the summary graph of G_o according to subset mapping $\mathcal{S}$.

The current research applies subset mappings where each original graph node in $\mathcal{V}_o$ is mapped to one subset in $\mathcal{Z}$. Therefore, M_s is a many-to-one mapping from graph nodes to a subset. However, if for $\mathcal{S}$ a mapping from a node to multiple subsets is created, M_s becomes a many-to-many mapping.

Definition 4. ***Summarization Function***
Let $G_o = (\mathcal{V}_o, \mathcal{E}_o, \mathcal{R}, \mathcal{T})$ be a graph (Definition 1). Let $\mathcal{S} \subseteq \mathcal{V}_o \times \mathcal{Z}$ be a subset mapping (Definition 3). Let $\mathcal{P}$ be a partition (Definition 2). We define the Summarization Function F which takes a graph G_o and a subset mapping $\mathcal{S}$ to produce a tuple:

$$F(G_o, \mathcal{S}) = (G_s, M_s)$$

$G_s = (\mathcal{V}_s, \mathcal{E}_s, \mathcal{R}, \mathcal{T})$ is a summary graph according to the subset mapping $\mathcal{S}$. The Summarization Function F creates a partition $\mathcal{P}$, according to the subset mapping $\mathcal{S}$ by allocating each original node in $\mathcal{V}_o$ to its mapped subset in $\mathcal{Z}$. We define the set of non-empty subsets in $\mathcal{Z}$ to be the partition $\mathcal{P}$. We define $\mathcal{V}_s$ to be the summary node set of G_S, where each node in $\mathcal{V}_s$ represents a subset in $\mathcal{P}$. $\mathcal{E}_s$ is the set of typed edges between the summary nodes in $\mathcal{V}_s$. Each edge in $\mathcal{E}_s$ corresponds to a relation between two summary nodes v_{si} and $v_{sj} \in \mathcal{V}_s$, denoted by $r\,(v_{si}, v_{sj})$, where $r \in \mathcal{R}$ is a relation type. We define M_s to be a mapping from each node in $\mathcal{V}_o$ to a summary node in $\mathcal{V}_s$.

3.3 Approximate Graph Summarization

In heterogeneous graphs, approximate graph summarization techniques create a partition (Definition 2) based on node characteristics like edge types or node types [8]. Nodes in the same subset of a partition are similar in some characteristic (but not all) making the subset of the partition an approximation for the nodes. Nodes in heterogeneous graphs can have multiple in- and outgoing edges with multiple edge types ($r \in \mathcal{R}$, Definition 1). The attribute summarization technique partitions on edge types, called attributes. The attribute summary relies on the semantic structure of the graph as it exploits the attribute set of each node to create a partition. The resulting attribute summaries differ in size and structure as a consequence of the differences in the subset mappings and the graph structure. In the remainder of this section we discuss three attribute subset mappings, relevant to this research. We define three attribute sets which encompass the scope of the subset mappings for the attribute summaries. We support each attribute summary with an example.

Attribute Summary The Incoming (In) attribute summary, Outgoing (Out) attribute summary and the Incoming/Outgoing (In/Out) attribute summary respectively partition on the incoming, outgoing and incoming/outgoing attributes. We formally define the incoming and outgoing attribute set with respect to a single node in Definition 5.

Definition 5. ***Attribute sets: Incoming & outgoing*** *(adapted from [4])*
Let $G = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$ be a graph (Definition 1).
We define $\mathcal{A}_{v\_in}$ to be the incoming attribute set of a node $v \in \mathcal{V}$ as:

$$\mathcal{A}_{v\_in} = \{r \in \mathcal{R} \mid r(v_p, v) \text{ where } v_p \in \mathcal{V} \text{ and } r \neq rdf : type\}$$

We define $\mathcal{A}_{v\_out}$ to be the outgoing attribute set of a node $v \in \mathcal{V}$ as:

$$\mathcal{A}_{v\_out} = \{r \in \mathcal{R} \mid r(v, v_o) \text{ where } v_o \in \mathcal{V} \text{ and } r \neq rdf : type\}$$

With attribute sets (Definition 5) we create subset mappings to be used in the Summarization Function (Definition 4). We formally define the subset mappings for the In attribute summary, Out attribute summary and the In/Out attribute summary in Definition 6.

Definition 6. ***Subset Mappings of Attribute Summaries*** *(derived from Definition 5 and [8])*
Let there be a graph $G_o = (\mathcal{V}_o, \mathcal{E}_o, \mathcal{R}, \mathcal{T})$. *Let* $F(G_o, \mathcal{S}) = (G_s, M_s)$ *(Definition 4). Let* $\mathcal{A}_{v\_out}$ *and* $\mathcal{A}_{v\_in}$ *be the outgoing and incoming attribute set of a node* $v \in \mathcal{V}_o$, *respectively (Definition 5).*

We call G_s *the Out attribute summary graph of* G_o *according to subset mapping* $\mathcal{S}_{out}$ *if:*

$$\mathcal{S}_{out} = \{u, x \in \mathcal{V}_o \times \mathcal{Z} \mid \mathcal{A}_{u\_out} = \mathcal{A}_{x\_out}\} \text{ and } F(G_o, \mathcal{S}_{out}) = (G_s, M_s)$$

We call G_s *the In attribute summary graph of* G_o *according to subset mapping* $\mathcal{S}_{in}$ *if:*

$$\mathcal{S}_{in} = \{u, x \in \mathcal{V}_o \times \mathcal{Z} \mid \mathcal{A}_{u\_in} = \mathcal{A}_{x\_in}\} \text{ and } F(G_o, \mathcal{S}_{in}) = (G_s, M_s)$$

We call G_s *the In/Out attribute summary graph of* G_o *according to subset mapping* $\mathcal{S}_{in/out}$ *if:*

$$\mathcal{S}_{in/out} = \{u, x \in \mathcal{V}_o \times \mathcal{Z} \mid \mathcal{A}_{u\_in} = \mathcal{A}_{x\_in} \wedge \mathcal{A}_{u\_out} = \mathcal{A}_{x\_out}\} \text{ and } F(G_o, \mathcal{S}_{in/out}) = (G_s, M_s)$$

Attribute Summaries Example We elaborate on the attribute summaries by discussing the example of the *Original Graph* in Figure 2. The original graph consists of nodes n_1 to n_4 and the directed edge set $\mathcal{R} = \{black, red\}$ represented as arrows. Below each graph of Figure 2, the attributes with its direction (*_in*, *_out*) for each node are indicated in a set. From the original graph we derive that the nodes $n1$ and $n2$ of the original graph have exactly the same incoming and outgoing attributes. However, nodes $n3$ and $n4$ have different attribute sets. The In attribute summary only takes the incoming attributes into account and neglects the outgoing edges, resulting in the attributes sets presented below the In attribute summary. Logically, $n1$ and $n2$ retain the same attribute set $\{black\_in\}$ as both nodes in the original graph have one incoming black relation. Node $n3$ retains two incoming attributes: *black_in* and *red_in*. Node $n4$ has no incoming attributes and retains an empty set. Considering the retained attribute sets, we see that $n1$ and $n2$ have a coinciding attribute set and can be mapped to the same summary node C. Node $n3$ is represented by summary node B and $n4$ by A. We construct the summary graph by substituting the original nodes with the summary nodes, respecting the edge connections between original nodes. For summary nodes that have an edge connected to a same summary node, we display the edge connected to itself (self loop). This results in the In attribute Summary, consisting of three nodes (Figure 2). The creation of the Out and the In/Out summaries follows the same steps. In Figure 13 (Appendix A) we provide an extended visualization of the creation of the In, Out and In/Out attribute summaries.

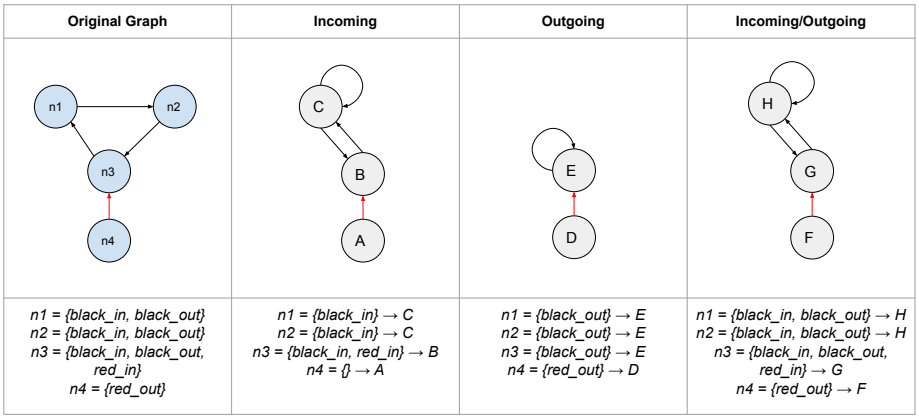

Figure 2. An example of an original graph and three resulting summary graphs created with the In attribute summary, Out attribute summary and the In/Out attribute summarizations. For each graph, the original nodes and their attribute set used for partitioning is presented. If the attribute sets of nodes coincide, it follows that the original nodes belong to the same subset of the partition. Nodes with an attribute empty set belong tot the same subset of the partition (node A of the In attribute Summary in this example).

3.4 Precise Graph Summarization

Approximate graph summarization partitions nodes on their local schema, while precise graph summaries have the ability to create structural summaries accounting for neighboring node characteristics over multiple hops [4, 10]. The quality of the precise summary graph relies evidently on how well the summary graph mirrors the original graph. Following the concept of homomorphism the structure of the original graph is preserved in a precise graph summary [8]. A graph summary is said to be precise if every path in the summary graph exists in the original graph. The inverse however, may not hold true as every path in the original graph does not necessarily have to exist in the summary graph. In order to define a precise graph summarization, we formally define a path set of existing (distinct) paths in the graph in Definition 7.

Definition 7. ***Path Set*** *(adapted from [8])*
Let $G = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$ *be a graph (Definition 1). Let* $p = (r(v_1, v_2) \wedge ... \wedge r(v_{n-1}, v_n))$ *where* $v_1, ..., v_n \in \mathcal{V}$ *and* $r \in \mathcal{R}$*, be an existing path in* G*. We define path set* $\mathcal{W}$ *as the set of distinct paths* p *in* G*:*

$$\mathcal{W} = \{(r(v_1, v_2) \wedge ... \wedge r(v_{n-1}, v_n)) \mid \ v_1, ..., v_n \in \mathcal{V} \ , \ r \in \mathcal{R}\}$$

With the path sets of both the graph and the summary graph, one can verify if a graph summary is precise by means of Definition 8. The precise graph summary definition states that a graph summary is precise if all paths in the

path set of the summary graph exist in the path set of the original graph after substituting the original nodes with its representing summary node with subset mapping $\mathcal{S}$ (Definition 4).

Definition 8. ***Precise Graph Summary*** *(adapted from [8])*
Let $G_o = (\mathcal{V}_o, \mathcal{E}_o, \mathcal{R}, \mathcal{T})$ be a graph (Definition 1). Let $F(G_o, \mathcal{S}) = (G_s, M_s)$ (Definition 4). Let $\mathcal{W}_o$ and $\mathcal{W}_s$ be the set of paths in G_o and G_s, respectively. G_s is a precise graph summary if:

$$\forall r((v_1, v_2), ..., r(v_{n-1}, v_n)) \in W_o:$$
$$(r(M_s(v_1), M_s(v_2)), ..., r(M_s(v_{n-1}), M_s(v_n))) \in W_s.$$

With mapping M_s every node $v_{ox} \in \mathcal{W}_o$ is replaced with a summary node $v_{sx} \in \mathcal{V}_s \in G_s$. G_s is a precise graph summary of G_o if all paths in W_s exist in W_o.

(k)-Forward Bisimulation The existence of a bisimulation, or equivalence relation, between parts of a graph implies that the parts are structural equivalent. Bisimulation is considered a binary relation and holds if two arbitrary nodes and their inverse are simulations. Furthermore, bisimulation is considered stronger if the nodes can be substituted by one another, making the bisimulation symmetric [8]. (k)-forward bisimulation captures bisimulation over k-hops. As *forward* implies, the forward, or outgoing, paths are considered. Over each hop, the forward path set has to be equivalent for nodes to be in the same equivalence class [4]. We formally define forward bisimulation in Definition 9.

Definition 9. ***Forward Bisimulation*** *(adapted from [8])*
Let $G_0 = (\mathcal{V}_o, \mathcal{E}_o, \mathcal{R}, \mathcal{T})$ be a graph (Definition 1). Let $\mathcal{W}_o$ be the path set of G_o (Definition 7). We define a forward bisimulation $\approx_{fwBisim}$ of two nodes in $\mathcal{V}_o$ as:

$$\forall r(x, x') \in \mathcal{W}_o \; \exists r(y, y') \in \mathcal{W}_o \; \wedge \; (x', y') \in \approx_{fwBisim}$$

and inverse,

$$\forall r(y, y') \in \mathcal{W}_o \; \exists r(x, x') \in \mathcal{W}_o \; \wedge \; (x', y') \in \approx_{fwBisim}$$

With the forward bisimulation $\sim_{fwBisim}$, we define the subset mapping of (k)-forward bisimulation, $\sim_{(k)-fwBisim}$, in Definition 10. The subset mappings consider equivalence classes, resulting from (k)-forward bisimulation. We define the (k)-forward bisimulation as the forward bisimulation over k-hops.

Definition 10. ***Subset Mapping of (k)-Forward Bisimulation Summary*** *(adapted from [8])*

Let G_o be a graph (Definition 1) and $\mathcal{W}_o$ be the path set (Definition 7) in G_o. Let $F(G_o, \mathcal{S}) = (G_s, M_s)$ (Definition 4). We define the (k)-forward bisimulation $\sim_{(k)-fwBisim}$ to be a forward bisimulation $\sim_{fwBisim}$ (Definition 9) over for k-hops, such that:

$$(\forall r(x, x') \in \mathcal{W}_o \; \exists r(y, y') \in \mathcal{W}_o \;\wedge\; (x', y'), ...,$$
$$\forall r(x_k, x'_k) \in \mathcal{W}_o \; \exists r(y_k, y'_k) \in \mathcal{W}_o \;\wedge\; (x'_k, y'_k) \in \approx_{fwBisim}) \;\in \approx_{(k)-fwBisim}$$

and inverse,

$$((\forall r(y, y') \in \mathcal{W}_o \; \exists r(x, x') \in \mathcal{W}_o \;\wedge\; (x', y'), ...,$$
$$\forall r(y_k, y'_k) \in \mathcal{W}_o \; \exists r(x_k, x'_k) \in \mathcal{W}_o \;\wedge\; (x'_k, y'_k)) \in \approx_{fwBisim}) \;\in \approx_{(k)-fwBisim}$$

We call G_s the (k)-forward bisimulation summary according to the subset mapping $\mathcal{S}_{(k)-fwBisim}$ if:

$$\mathcal{S}_{(k)-fwBisim} = \{u, x \in \mathcal{V}_o \times \mathcal{Z} \mid u, x \in \approx_{(k)-fwBisim}\} \text{ and,}$$
$$F(G_o, \mathcal{S}_{(k)-fwBisim}) = (G_s, M_s)$$

Notably, increasing k in the (k)-forward bisimulation summarization generally results in more distinct equivalence classes in heterogeneous graphs. Less nodes will be bisimilar, as over more hops forward bisimulation should hold for nodes to be in the same equivalence class. It is likely that when k in (k)-forward bisimulation is increased, the summary graph size increases. This may jeopardize the initial purpose of the (k)-forward bisimulation summarization.

(k)-forward Bisimulation Example The (k)-forward bisimulation summarization relies on iterative message passing. The messages contain forward path label sets. Messages are passed in the inverse direction of the forward paths for k-iterations. In Figure 3 an example of the (2)-forward bisimulation is displayed. Two fragments, *X* and *Y*, of the same graph are visualized. In the first iteration (k=1), the massage to be passed is the local forward path set of each node. Nodes *n4, n5, n98* and *n99* do not posses a forward path. After the first message passing, nodes *n1* and *n95* contain the path set {*black_out, red_out*}. Nodes n2 and n96 contain the path set {*orange_out*}. Nodes n3 and n97 contain the path set {*blue_out*}. In the second iteration (k=2), message passing consists of passing the collected path sets from the first iteration (k=1). Logically, nodes *n4, n5, n98* and *n99* pas the empty set. Nodes *n1* and *n95* cannot pass the collected paths, as the nodes are not connected in this example. *n1* and *n95* receive the messages from *n2, n3* and *n96, n97*, respectively. The resulting path set for *n1* and *n95* is now $\{\{black\_out, \{orange\_out\}\}, \{red\_out, \{blue\_out\}\}\}$. Note that, the messages passed form *n2* to *n1* is join the path subset of *black_out*. This way, precise graph summarization is accomplished, as all paths of the summary graph should exists in the original graph. after k-iterations of message passing, each equivalence class is a subset in the partition. The (2)-forward bisimulation

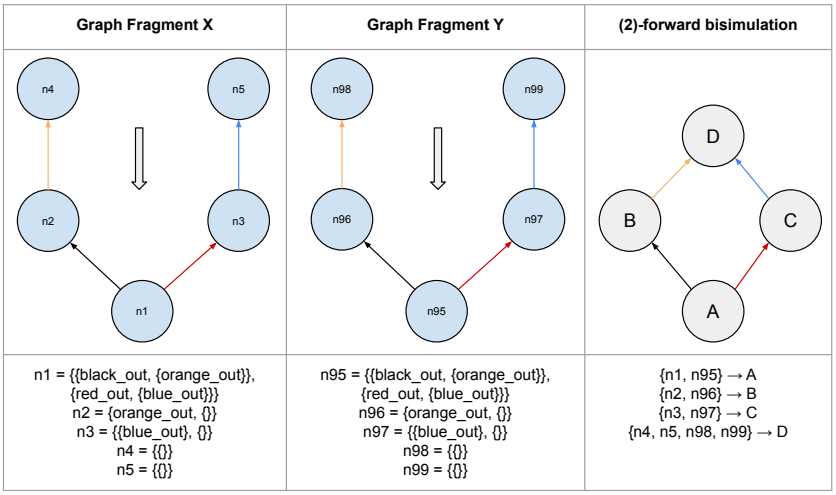

Figure 3. (2)-forward bisimulation example. X and Y represent graph fragments. For each node the forward path sets are collected over 2-hops. Below each graph fragment the paths sets over 2-hops are presented. Partitioning is carried out considering equivalence classes.

summary is created by respecting the equivalence between original graph nodes, based on the collected forward paths over k-hops (Figure 3).

Remark 2. The (k)-forward bisimulation summarization with parameter k=1 is the same as the Out attribute summarization. (1)-forward bisimulation partitions on equivalence classes that are created with the forward paths over 1-hop. This is the same as partitioning on the outgoing attribute.

4 Relational Graph Convolutional Network

In this section the R-GCN model is discussed. First, we explain the message passing framework of the Graph Convolution Network (GCN). The R-GCN model's core operation is comparable to the message passing framework of the GCN model.

The GCN model learns the vector representations of the entities in the graph. The mechanism for GCN relies on the message passing framework. The message passing framework can be explained as a matrix multiplication, where the message passing for a single GCN layer in directed graphs is [40]:

$$H = \sigma(AXW) \tag{1}$$

Here X is the entity feature matrix. The Weights are denoted by W. σ is a non-linearity and A is a normalized Laplacian adjacency matrix of the graph. The feature matrix X indicates presence and absence of features. The operation of equation 1 is called message passing as the information of neighboring entities is passed to update every entity. Looking at a single node update with GCN, we rewrite Equation 1 as [40]:

$$h_i = \sigma\left[\sum_{x \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} \mathbf{x}_i^T W\right] \tag{2}$$

The output vector h_i is the updated representation for node i. N_i are the representations of incoming edge neighbors. N_i is used to calculate an average of the sum of vector representations of the neighbors of i. The average is multiplied by the current representation $\mathbf{x_i}$ of the to-be-updated node i and a weight matrix W. Then, a non-linearity σ is applied. Node h_i updated is constructed from neighboring vector embeddings and the previous node embedding of i.

The R-GCN model extends GCN to learn node representations on large multi-relational graphs [31]. R-GCN accounts for different relations and the directions of edges (incoming and outgoing) for node representation updates. The message passing operation of a single R-GCN layer with multiple relations can be derived from Equation 1 as [40]:

$$H = \sigma\left(\sum_{r=1}^{R} A_r X W_r\right) \tag{3}$$

Here, the adjacency matrix A_r describes the edge connections between the nodes. For each relation in the graph, $r \in \mathcal{R}$, there exists a relation specific weight matrix W_r. The message passing architecture enables training of the model with back propagation. The update for a single entity v_i is derived from the massage passing framework of the R-GCN layer (Equation 3) as [31]:

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right) \tag{4}$$

$\mathcal{N}_i$ is the set of neighboring nodes connected via incoming and outgoing edges. W_r is the relation specific weight and h_j is a neighboring node. The sum of the node vector representations of each neighbor multiplied by the relation specific weight of the connected edge is taken into account. Furthermore, W_0 is a special weight added to each node that functions as a self-loop. Node i is updated by taking the neighboring node representations into account as well as the current representation of i itself. Therefore, stacking two R-GCN layers, the node representation at layer l is taken into account for updating the same node at layer $l+1$. $c_{i,r}$ is a regularization term which can be modified according to the desired implementation [31].

5 Methodology

In the following section we elaborate on the methods for scaling R-GCN training with graph summaries and entity embedding transfer. First, we discuss graph data is processing. Then, we elaborate on how the entity embedding for the original graph is created and transferred. We conclude the chapter by providing an overview of the pipelines of the embedding transfer models and the *baseline* model.

5.1 Graph Processing

In order to execute summary graph training, graph summaries are created with the attribute summarization and with the (k)-forward bisimulation for each graph. The attribute summaries are constructed with the In, Out and In/Out attribute summarization technique. The attribute summarizations performed in this research are inspired by related work[8]. (k)-forward bisimulation summarization is performed using the FLUID framework from [4]. By increasing the parameter k=1 to k=3 in (k)-forward bisimulation, three summary graphs are obtained for each graph dataset. For the AM dataset we did not create an (3)-forward bisimulation due to computational limits. In Appendix B details of the graph datasets and the graph summaries are presented.

Applying the attribute and the (k)-forward bisimulation summarizations, the *rdf:type* relation is excluded from the attribute set and are not accounted for when creating the graph summaries. A special type of node is the literal node. Literal nodes contain unstructured data like text, numerical values and images. Literal nodes have one incoming edge and no outgoing edge and contain information about the incoming edge node. Each literal node is substituted with the same literal URI and are taken into account accordingly when summarizing. Another distinct node is the blank node. A blank node, is a node with an anonymous resource. It therefore has no Uniform Resource Identifier (URI)[3]. Blank nodes do have a structural function in graph as at connects with other nodes

[3] The URI is a sequence of characters that represents a logical or physical source of the node or relation in the graph.

just as any node with a URI. From data exploration it was concluded that blank nodes also occur having an *rdf:type* relation. blank nodes are summarized and processed like any other regular node. Triples that contain the Web Ontology Language predicated are removed as suggested in related work [8].

Graph Labels The entity type prediction task in this research relies on the *rdf:type* relation of the entities in the graph. It should be noted that this is a different task than presented in related literature performs [31, 40]. The labeled nodes in the train and test data in these latter papers rely on class relations specific to the graph datasets. literature [3] points out that the resulting training and test split consist of few instances. The authors of the paper state that small test and training dataset could ambiguously reflect the performance of the model. For the experiments in the current research the *rdf:type* relations are pruned from the graph and from the relation set $\mathcal{R}$ (Definition 1). This prevents that the type labels and the *rdf:type* relation are trained. The *rdf:type* relation is used to create class labels for each node in the graph and summary graphs. For the original graph, class labels are created by extracting the *rdf:type* relation. An example is provided in Table 1. Four graphs nodes an their *rdf:type* labels are indicated by '1'. As Table 1 shows, 'node3' has two type labels. These labels are vectorized for training on the original graph.

		Labels		
Node	Relation	Male	Female	Child
node1	*rdf:type*	[1,	0,	0]
node2	*rdf:type*	[1,	0,	0]
node3	*rdf:type*	[0,	1,	1]
node4	*rdf:type*	[1,	0,	1]

Table 1. Example of graph nodes with type labels denoted by '1' for being of that type and '0' for not being of that type. In this example a node can have multiple type labels indicating a multi-label classification problem.

Similar node labels are created for the summary graph nodes. Mapping M_s (Definition 4) is used to produce weighted labels for the summary graph nodes. The label of a summary node is calculated by counting the occurrence of each type label of the original nodes that are mapped to the summary node. the count for each label is divided by the number of nodes mapped to the particular subset, creating the weighted label. We provide an example in Table 2, where summary graph nodes and original graph nodes are presented. *s_nodeA* is the summary node of *node1* and *node2*. *s_nodeB* is the summary node of *node3* and *node4*. The summary labels are a weighted representation of the original binary labels of the original node labels. In Table 2 the weighted labels are displayed in vectorized form for *s_nodeA* and *s_nodeB*.

			Summary Labels		
Summary Node	Relation	Node	Male	Female	Child
s_nodeA	*summaryOf*	node1, node2	[1,	0,	0]
s_nodeB	*summaryOf*	node3, node4	[1/2,	1/2,	1]

Table 2. Weighted type labels for summary nodes A and B (s_nodeA, s_nodeB) created by count each type label of the original nodes in the partition and dividing each label that has a count higher than 0 by the number of nodes that are mapped to the summary node.

5.2 Entity Embedding Transfer

The contribution of this research lies in the entity embedding transfer of the learned embeddings with summary graph training. The entity embedding for each summary graph has the shape (s, d), where s is the number of summary graph entities and d is the embedding dimension. During training on the summary graph the embeddings will be updated through backpropagation. After training on the summary graph there is one embedding for each summary graph. each original node is represented by an summary node, which can be retrieved by using mapping M_s (Definition 4). With one of the below described methods the embedding for the original graph is constructed and transferred.

Summation For the embedding transfer with the *summation* model first a list of tensors is created. The list of tensors consist of tensors of the shape (n, d), where n is the number of original graph nodes. The number of tensors in the list is equivalent to the number of summary graphs which are used for pre-training. Each tensor in the list presents one summary graph. For each node in the original graph, the representing summary node is located with mapping M_s. Then, the embedding of the summary node is copied to the index of the original node in the particular embedding tensor the list. We carry out this process for each original graph node and each summary graph embedding. Now, we have a list of embedding tensors. The tensor list is summed in to one tensor by summing the embeddings for each index in across the tensors in the. The result is one entity embedding for the original graph of shape (n, d).

Multi-Layer Perceptron For the embedding transfer with the multi-layer perceptron (*mlp*), a concatenated embedding tensor is constructed. The first step is the same is in the *summation* model. After creating the tensor list with copied summary node embeddings, the tensor list is concatenated into one tensor. The result is a tensor of shape $(n, x \times d)$, where x is the number of graph summaries. The concatenated tensor is fed to the *mlp* model consisting of two linear layers. The second layer of the *mlp* outputs an original graph embedding of shape (n, d).

Multi-Head Attention The multi-head attention implemented in this research is proposed in [42]. Multiple attention heads enable for diverse attention to be paid to different elements. Each head represents a separate attention mechanism. The multi-head attention module cycles simultaneously through multiple attention mechanism. The predicted output is created by linearly combining the separate attention outputs. Multi-head attention is able to receive input of multiple features of the same object. In this case, the multi-head attention layer receives multiple summary node embeddings that can be considered as features for the original node embedding.

For transferring the summary graph embeddings with multi-head attention, like in the *summation* method a tensor list is created. The tensors in the tensor list are then stacked, yielding a 3-dimensional tensor of shape (x, n, d). The multi-head attention processes a Query (Q), Key (K) and Value (V) of the same shape. The stacked tensor is provided as Q, K and V to the multi-head attention. The output is an attended tensor of the same shape (x, n, d). We use the tensor at index 0 as input for the R-GCN layers.

5.3 Model Pipelines

The entity and R-GCN weight transfer models have similar pipelines. The models differ in the embedding transfer method. The steps in the pipeline for each transfer model are:

1. Graph processing: graph processing involves creating the graph summaries and creating node labels to perform the entity type prediction task for summary graph and original graph training.
2. Summary graph training: summary node embeddings and R-GCN weights are trained. During summary graph training the relation specific weights of the R-GCN model are shared. Summary graph training yields as many embeddings as there are summary graphs.
3. Embedding and R-GCN weights transfer: the pre-trained embeddings are processed into a new embedding that fits the original graph. With summation, a multi-layer perceptron or a multi-Head attention layer, a new node embedding is constructed and transferred together with pre-trained R-GCN weights.
4. Original graph training: the original graph and the constructed entity embedding are input for a new R-GCN model. The new R-GCN model, consists of two R-GCN layers. The two R-GCN layers are initialized with the pre-trained R-GCN weights.

A schematic overview of the pipelines for each model, after the graph processing step, is displayed in Figure 4. Most left in Figure 4, summary graph training is displayed. The summary graph training with three summary graphs produces three entity embeddings. The entity embeddings are transferred to the *summation*, *mlp* or *attention* model. The embeddings are processed according to the model specific method. In original graph training (right in Figure 4) the

constructed entity embedding and the original graph are input for the R-GCN layers. These R-GCN layers are initialized with R-GCN weights, obtained with summary graph training.

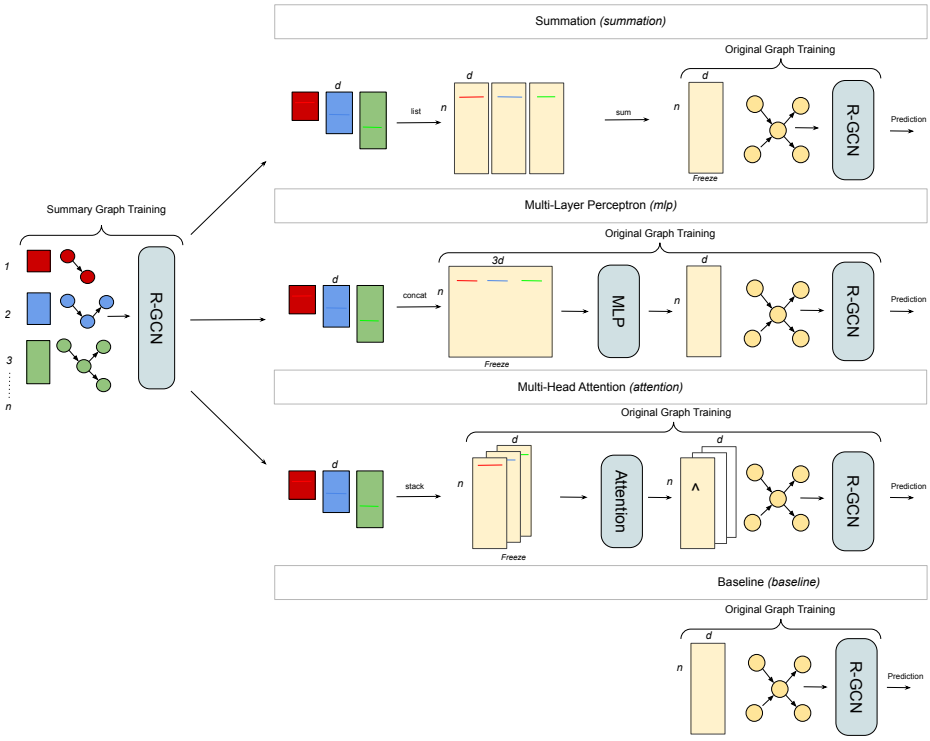

Figure 4. The figure presents pipelines for the different models. In this particular case, three summary graphs are fed to the R-GCN model, resulting in three node embedding tensors and pre-trained R-GCN weights. In the transfer model pipelines the embedding tensors are used to create a new node embedding. The new embedding and the original graph are input for original graph training. The R-GCN layers for original graph training are initialized with pre-trained R-GCN weights, obtained with summary graph training. The model names correspond to the embedding transfer method.

6 Experimental Setup

This chapter presents the experimental setup for evaluating the transfer models on scaling of R-GCN training. All experiments are carried out on CPU[4]. The transfer models proposed in this research are evaluated on the widely used graph datasets AIFB[2], MUTAG[13] and AM[5]. Full batch (whole graph) training is conducted for summary graph and original graph training as proposed previously[31, 40]. For each graph dataset a 60%/20%/20% split for training/validation/testing, respectively, is created.

In the remainder of this section we discuss the activation and loss functions we use for the transfer models. Then, we discuss metrics to evaluate our models during run time and on the test sets. After that, hyper-parameters for the transfer models and baseline model are discussed. At last, we present a table with elements of the transfer models and the baseline models.

6.1 Activation & Loss

For the R-GCN layers in the transfer and baseline models, a nearly same setup is used for summary graph and original graph training. The R-GCN setups differ in activation function over the R-GCN output layer and loss computation. The models consist of a 2-layer R-GCN with a ReLU activation on the output of the first layer. On the output of the second R-GCN layer a Sigmoid or Softmax activation is applied. In original graph training, the models apply the Sigmoid and the Softmax activation over the R-GCN output layer when multi-label and multi-class classification task, respectively, are performed.

The graphs used for the experiments contain nodes that could have multiple labels, making it a multi-label classification problem. If a node only contains one label of the multiple classes, we call it a multi-class classification problem. In Table 9 in Appendix B, we provide details about the graph datasets that we use for evaluation. The AIFB dataset contains over 50% multi-labeled nodes. MUTAG contains no multi-labeled nodes and the AM dataset contains three multi-labeled nodes. Three multi-labeled nodes is negligible small compared to the single labeled nodes in the AM dataset (Table 9 in Appendix B). For the three labeled nodes, we randomly assign only one of the labels. The AIFB dataset and summary graph training is considered to be a multi-label classification problem, as nodes could have multiple type labels.

In the case that multiple labels can be predicted, the Sigmoid activation is desired as we want to make a prediction for each label independently. We do not only predict the most likely label, as with the Softmax activation. Binary Cross Entropy Loss (BCELoss) is a suitable loss calculation for multi-label classification combined with a Sigmoid activation on the output layer. The BCELoss is applied as it calculates the loss over each prediction and label value separately, treating each output independent. BCELoss sets up a classification problem for each class. For multi-class classification (MUTAG and AM dataset) our setup

[4] 96 GB Intel Xeon Silver 4110 2.10 GHz processor

is in line with related work [40]. Only one label for each entity is predicted in multi-class classification. Therefore, a Softmax activation over the R-GCN output is applied. We compute the loss with the Categorical Cross-Entropy Loss function when we apply the Softmax activation in the multi-class classification.

For the multi-layer perceptron in the *mlp* model we use a TanH activation on the first layer. Generally, the TanH activation accounts for larger gradients and thus larger updates in the weights and entity embeddings. Also, using TanH can contribute to a faster convergence. Both characteristics of the TanH activation of interest when scaling R-GCN training. On the second linear layer of the multi-layer perceptron we do not apply an activation, as this output is the newly constructed entity embedding.

6.2 Metrics

Data analysis on the graph datasets showed class imbalance in typed entities. In the MUTAG dataset for example, some entity types only occur two or three times. We decided to include the less occurring entity types from prediction as we think this a homogeneity characteristic and could also occur in larger graphs or unstructured web data. To account for the imbalanced data we evaluate model performance not only on accuracy, but also on F1 weighted and on F1 macro. The accuracy is calculated as in related work[40]. As the entity type prediction task concerns multiple classes, the accuracy is computed for every subset of predicted labels. The predicted set of labels for a sample must exactly match the corresponding set of true labels the achieve 100% accuracy. Predicting that an entity is not of a certain type is accounted for in the accuracy calculation. The F1 weighted and F1 macro scores rely on the F1 computation, presented in Equation 5.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

The F1 weighted score is calculated by taking the mean of all F1 scores per class type while considering the support of each class. The F1 weighted score treats classes unequally according to their support. The F1 macro score is computed using the unweighted mean. The mean of the summed F1 scores per class is calculated without accounting for the class support. The F1 macro score treats all classes equally regardless of their support values.

6.3 Hyper-Parameters

In Table 3 we provide hyper-parameters for the experiments and our models. For each experiment holds that we iterate the experiment 5 times. For each summary graph we train for 51 epochs. Then, we train on the original graph for 51 epochs. In each epoch during original graph training, we first evaluate the model on the validation set. After 51 training epochs on the original graph, the models are evaluated on the test set. During, validation runs we found that an embedding

tensor with dimension $d = 63$ performed better than the proposed $d = 500$ in related work [40]. Aside from the embedding dimension, we do not perform any parameter tuning and implement R-GCN settings found in related work [31, 40]. In summary graph training and original graph training we use a hidden layer of 16 units between the two R-GCN layers. l2 regularization of 5×10^{-4} is applied on the two R-GCN layers. For the *mlp* model we adjust the output of the first layer according the size of the embedding dimension. Remember, we create a concatenated tensor as input for the *mlp*. The hidden units, in the hidden layer in the *mlp*, are calculated by $\frac{2xd}{3} + c$, where x is the number of graph summaries, d the embedding dimension and c the number of prediction classes. For the multi-head attention layer we set the number of attention heads to be equal to the amount of graph summaries (x) that we train use in summary graph training. Dropout in the multi-head attention layer is set to 2×10^{-1}. We use Kaiming He initialization for all weights except for the entity embedding which is initialized with the standard normal distribution $\mathcal{N}(0, 1)$. The Adam optimizer with a learning rate of 10^{-2} is used for summary graph and original graph training in each model.

Parameter	**Value**
Epochs	51
Iterations	5
Learning Rate	10^{-2}
Weight Decay	5×10^{-4}
Embedding Dimension	63
R-GCN Hidden Units	16
mlp Hidden Units	$\frac{2xd}{3} + c$
attention Heads	x
attention Dropout	2×10^{-1}

Table 3. Hyper-parameters of the models during experimentation. x is the number of graph summaries and d is the embedding dimension.

6.4 Models Overview

An overview of the transfer models is displayed in Table 4. The specific module layers for each model in summary graph training and original graph training are indicated. Summary graph training is conducted with the same setup in each model (Table 4). The RGCNConv[31], Embedding, Linear and Multi-Head Attention[42] modules are provided by PyTorch. Note that, the *baseline* model does not contain a summary graph training module and only conducts original graph training. Table 4 presents the discussed activation functions (section 6.1) for each layer. For the *mlp* and the *attention* models the multi-layer perceptron and multi-head attention layer, respectively, are displayed in the original graph layers column. The column *Parameters (AIFB)* denotes the trainable parameters

for each model including the parameters of the *Summary Graph Layers*. The model parameters were calculated based on training with the AIFB graph and three attribute summaries (In, Out and In/Out), where the embeddings are frozen (no gradients) after transfer. By default the embedding tensors are frozen after the embedding transfer. Freezing the transferred node embedding reduces the amount trainable parameters: the original graph embedding is not trained, but constructed from summary graph embeddings, which are smaller than the original graph node embedding. In the model parameter calculation, the R-GCN weights of the original graph layers contain gradients.

Model Name	**Summary Graph Layers**	**Embedding Trick**	**Original Graph Layers**	**Parameters (AIFB)**
summation	Embedding RGCNConv (ReLU) RGCNConv (Sigmoid)	Summation	Embedding RGCNConv (ReLU) RGCNConv (Sigmoid/Softmax)	116566
mlp	Embedding RGCNConv (ReLU) RGCNConv (Sigmoid)	Concatenation	Linear (TanH) Linear (None) RGCNConv (ReLU) RGCNConv (Sigmoid/Softmax)	195059
attention	Embedding RGCNConv (ReLU) RGCNConv (Sigmoid)	Stacking	MultiHeadAttention RGCNConv (ReLU) RGCNConv (Sigmoid/Softmax)	132694
baseline	None	None	RGCNConv (ReLU) RGCNConv (Sigmoid/Softmax)	584152

Table 4. Overview of the models with different embedding transfers. For each model the layer modules and activation functions are indicated. The summary graph training always has the same set up in each model. The model parameters are calculated on the AIFB dataset with three attribute summaries. The last column of table displays that the transfer models contain less trainable parameters than the *baseline*.

7 Experiments & Results

In this section we present experiments to answer our research questions. We analyze the results of each experiment after describing the experiment. Our findings are supported with tables and figures. Note that we iterate every experiment 5 times (Table 3). In the figures we always present metric curves that represent the average value of the sample. The standard deviation of the sample is displayed by a same colored area around the mean.

7.1 Multiple Summary Graphs Experiments

How can multiple graph summaries scale R-GCN training for entity type prediction in graphs?

With this experiment, we aim investigate how multiple graph summaries can scale R-GCN training. Summary graph training is conducted with the In, Out and In/Out attribute summaries. For (k)-forward bisimulation the k=1, k=2 and k=3 summary graphs are used for summary graph training. After summary graph training the R-GCN weights and node embeddings are transferred by the transfer models. We track run time during training of the *summation* and *baseline* model on the AIFB dataset where we include the creation of the attribute summaries and summary graph training in the run time for the *summation* model. We do not carry out the run time experiment with (k)-forward bisimulation because the creation of these summaries is computationally intensive and is relatively time consuming. After the embedding trick (Table 4) we freeze the node embeddings. We feed the constructed embedding to the model and is not updated by backprogation during original graph training. Note that, the R-GCN weights and the weights of the *mlp* and *attention* model do have gradients. We evaluate the models on validation accuracy and loss during original graph training. On the test sets, we report the accuracy, F1 weighted and F1 macro scores.

Multiple Summary Graphs Experiments: Results In Figures 5 and 6 we display the average AIFB validation accuracy during training epochs, of the *summation* and *attention* models, respectively. The *summation* and *attention* model are initialized with embedding and R-GCN weight transfer. The pre-trained embedding and R-GCN weights resulted from summary graph training on the attribute summary set. What notice that the *summation* model predicts with an higher accuracy on the validation set than the *attention* and *baseline* model in epoch 0. The *attention* model intersects the y-axis at 0.01 (1.24%) and the *baseline* model at 0.00 (0.0%). The curve of the *summation* model intersects the y-axis at 0.39 (39.11%). This is an expected result since the *summation* model does not contain any trainable parameters, besides the R-GCN weights. The attention weights of the *attention* model explain the lower y-axis intersection as the attention weights are not trained yet. The prediction at epoch 0 of the

attention model is comparable to the *baseline* model. Both curves of the *summation* model (Figure 5) and *attention* model (Figure 6) show a steep increase in the validation accuracy in earlier training epochs compared to the *baseline*, indicating scaling of R-GCN training.

For the *summation* and *baseline* model, run time was measured on the AIFB dataset (Figure 5). The run time was measured, including attribute summary graph creation, summary graph training and original graph training. We found that the *baseline* model predicts on average with a highest accuracy of 74.00% after epoch 40 on the AIFB validation set. The run time at this point for the *baseline* model is 15 seconds. After 15 seconds of run time, including the creation of the attribute summaries, the *summation* model predicts with a validation accuracy of 88.89% on the AIFB dataset in epoch 15.

We analyzed the AIFB validation accuracy curve during training epochs of the *summation*, *mlp* and *attention* models that conduct summary graph training on the (k)-forward bisimulation set. We observe similar accuracy curves as for the *summation* and *attention* model that trained on the attribute summary graph set (Figures 5, 6). The *mlp* model was the best performing model with the (k)-forward bisimulation summary graph training. The validation accuracy curve of the *mlp* model with the (k)-forward bisimulation summary graph training is displayed in Figure 14 (Appendix C).

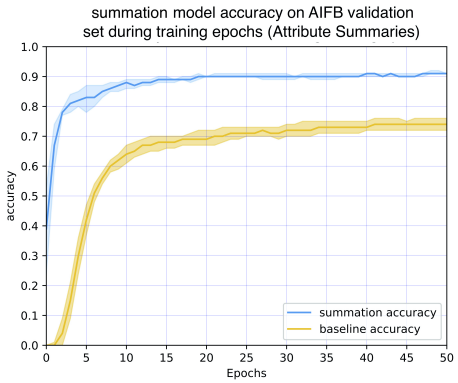

Figure 5. *summation* and *baseline* model mean accuracy on the AIFB validation set. The *summation* model predicts with an accuracy of 39.11% in epoch 0. The *summation* model has a steeper learning curve in early epochs and predicts with a significant higher accuracy after 51 training epochs than the *baseline* model.

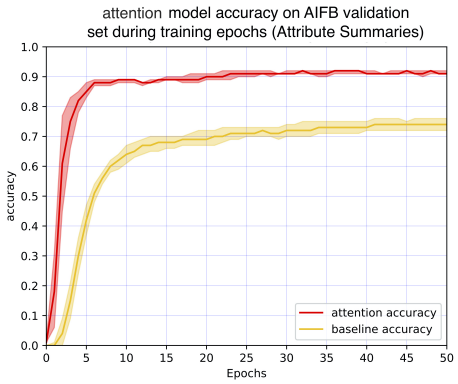

Figure 6. *Attention* and *baseline* model accuracy on the AIFB validation set. The *attention* model predicts with a 1.24% in epoch 0 on the validation set. The *attention* model has a steeper learning curve in early epochs and predicts with a significant higher accuracy after 51 training epochs than the *baseline* model.

Figure 7 and Figure 8 display the accuracy and the loss, respectively, on the MUTAG validation set for the *attention* and the *baseline* models. The *attention*

model conducted summary graph training on the attribute summary set. In Figure 7 we observe a steeper accuracy curve in early epochs for the *attention* model compared to the *baseline*, indicating scaling of R-GCN training. We do notice a accuracy drop with higher standard deviation around epoch 4. From epoch 5 and further it seems that the *attention* model converges. The *baseline* model seems to converge in epoch 28. The cross entropy loss curve (Figure 8) of the *attention* model suggest that the *attention* model has converged. In the loss curve of the *attention* model we measure a steep decline in early epochs, corresponding to the steep increase in validation accuracy in early epochs. Furthermore, A lower intersection with the y-axis for the *attention* loss curve compared to the *baseline* loss curve is measured. The *baseline* loss curve seems to indicate that the model has not converged after 51 training epochs (Figure 8). The accuracy result on the validation set during training epochs suggests that the *attention* model with the embedding transfer scales training for the MUTAG dataset as it displays as steeper curve in early epochs.

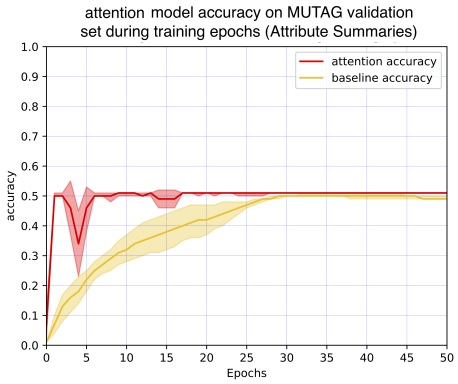

Figure 7. *attention* and *baseline* model accuracy on the MUTAG validation set. The *attention* model predicts with an accuracy of 0.33% in epoch 0. The *attention* model has a steeper learning curve in early epochs. The *attention* model predicts with a comparable accuracy on the validation set in epoch 51.

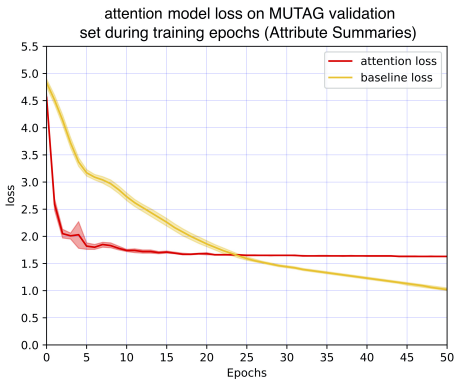

Figure 8. *attention* and *baseline* model loss on the MUTAG validation set. A steep decline in loss can be reviewed for the *attention* model in early epochs. The loss curve of the *attention* model seems to indicate that the model has converged. The *baseline* model loss curve seems not to be converged after epoch 51.

Validation accuracy curves of the *summation*, *mlp* and *attention* model on the AM dataset show similar results compared to the AIFB dataset curves. The Figures 9 and 10 display the average accuracy curves of the *mlp* and *baseline* models on the AM validation set during training epochs. The accuracy curves of the *mlp* model, trained on the attribute summaries (Figure 9) and (k)-forward bisimulation summaries (Figure 10), show scaling of R-GCN training. The accuracy curves of the *mlp* model trained on the attribute summaries shows to

have a higher standard deviation between epoch 3 and 5, compared to the *mlp* model trained on (k)-forward bisimulation summaries. Figure 11 displays the average *attention* and *baseline* accuracy curves on the AM validation set. We measured a higher standard deviation of the mean validation accuracy of the *attention* model between epoch 2 and 6. For the *summation* model we observe R-GCN scaling on the AM validation with both the attribute and (k)-forward bisimulation summaries (Figures 16, 17, Appendix C). The *summation*, *mlp* and *attention* models predict with a significant higher accuracy on the AM validation set after 51 training epochs compared to the *baseline* (Figures 9, 10, 11, 16, 17).

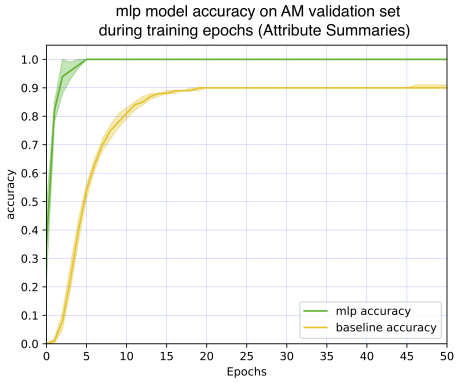

Figure 9. *mlp* and *baseline* model accuracy on the AM validation set. The *mlp* model conducted summary graph training with attribute summaries. A steeper increase of the prediction accuracy of the *mlp* model in early epochs is observed compared to the *baseline*. The *mlp* model predicts with a higher accuracy on the validation set at epoch 51 than the *baseline* model.

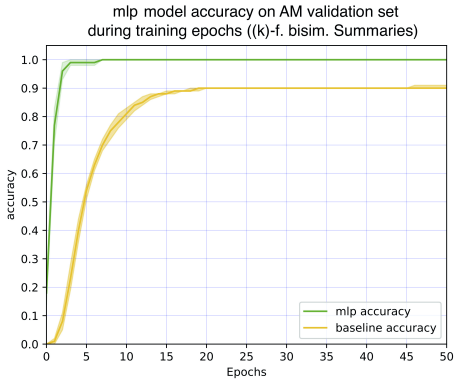

Figure 10. *mlp* and *baseline* model accuracy on the AM validation set. The *mlp* model conducted summary graph training with (k)-forward bisimulation summaries. A steeper increase of the prediction accuracy of the *mlp* model in early epochs is observed compared to the *baseline*. The *mlp* model predicts with a higher accuracy on the validation set at epoch 51 than the *baseline* model.

After training epochs, the models are evaluated on the test set measuring the accuracy, F1 weighted and F1 macro. The results are provided in Table 5. A general observation is that the F1 macro results are relatively low compared to the accuracy and F1 weighted scores. This was to be expected due to the class imbalance in the datasets. The F1 weighted and F1 macro results of the AIFB and AM datasets indicate that the models learn to predict on the type classes that have a larger support in the training dataset. The F1 weighted scores for the MUTAG dataset suggest that that there is significant support for class types that are predicted with low recall and precision. Analyzing the raw prediction results for the MUTAG test set, we notice indeed that despite the support of some classes, the model is not able to predict well on these classes. Therefore, accounting for the support, the F1 weighted score will be affected.

In Table 5 we observe that the transfer models outperform the *baseline* on accuracy, F1 weighted and F1 macro in the experiments on all datasets. For the AIFB and AM dataset the test set results of the models that pre-train on attribute summaries seem to outperform models that pre-train on the (k)-forward bisimulation summaries. For the AIFB dataset the highest prediction accuracy of 92.53% was yield by the *attention* model with attribute summary graph pre-training. The validation accuracy during training epochs of this *attention* model is presented in Figure 6. The mlp model in combination with the (k)-forward bisimulation summaries, predicted with 90.37% accuracy on the AIFB test set (accuracy curve in Figure 14). It seems that the (k)-forward bisimulation summary graph training in combination with the *attention* model has the best influence on the MUTAG test set prediction. The *attention* model that used (k)-forward bisimulation summary graph training, yields a 52.39% accuracy on the MUTAG test set. The validation accuracy of this model during training epochs is displayed in Figure 7. On the AM dataset, the *mlp* model that used attribute summary graph training had the best accuracy performance on the test set of 99.99%. The validation accuracy on the AM dataset of this *mlp* model is displayed in Figure 9.

	AIFB			**MUTAG**			**AM**		
	Acc	$F1_w$	$F1_m$	**Acc**	$F1_w$	$F1_m$	**Acc**	$F1_w$	$F1_m$
baseline	72.85 ± 1.05	84.15 ± 0.48	40.27 ± 1.70	48.67 ± 0.83	32.71 ± 0.64	3.92 ± 0.44	90.48 ± 0.18	93.07 ± 0.03	72.32 ± 0.05
attribute									
summation	92.14 ± 0.54	95.44 ± 0.49	63.49 ± 1.4	50.19 ± 0.21	37. 82 ± 0.22	4.25 ± 0.23	99.97 ± 0.04	99.97 ± 0.04	76.08 ± 0.15
mlp	91.19 ± 0.59	94.95 ± 0.45	62.25 ± 1.2	50.12 ± 0.19	37.67 ± 0.20	4.17 ± 0.35	**99.99 ± 0.01**	**99.99 ± 0.01**	76.15 ± 0.02
attention	**92.53 ± 0.40**	**95.49 ± 0.24**	**63.49 ± 0.35**	50.33 ± 0.21	37.84 ± 0.20	4.67 ± 0.35	99.92 ± 0.05	99.92 ± 0.05	75.95 ± 0.20
(k)-f. bisim.									
summation	89.13 ± 0.84	93.42 ± 0.35	58.99 ± 1.60	52.31 ± 0.15	39.71 ± 0.15	**5.91 ± 0.49**	99.91 ± 0.01	99.91 ± 0.01	76.13 ± 0.01
mlp	90.37 ± 0.58	94.01 ± 0.39	60.36 ± 0.75	52.25 ± 0.17	39.55 ± 0.27	5.82 ± 0.66	99.96 ± 0.01	99.96 ± 0.01	**76.16** ± 0.01
attention	88.53 ± 0.56	93.12 ± 0.39	56.87 ± 1.61	**52.39 ± 0.15**	**39.75 ± 0.11**	5.56 ± 0.23	99.70 ± 0.12	99.70 ± 0.12	75.80 ± 0.18

Table 5. Test set results of R-GCN scaling with multiple summary graphs. Accuracy, F1 weighted and F1 macro of the transfer models and *baseline* model on the AIFB, MUTAG and AM test sets. The *baseline* model trains only on the original graph. The Transfer models were initialized with a node embedding and R-GCN weights yield with summary graph training. Summary graph training occurred on three attributes summaries or on three (k)-forward bisimulation summaries. After the embedding and R-GCN weight transfer, the transfer models train on the original graph. Remarkable is that the transfer models outperform the *baseline* model on each test set on accuracy, F1 weighted and F1 macro.

7.2 Single Summary Graph Experiments

How does a single graph summary influence scaling of R-GCN training and R-GCN performance on entity type prediction? We investigate if R-GCN training can be scaled with a single summary graph, created with the attribute and (k)-forward bisimulation summarizations. The following experiment extends the results of previous research [14] by conducting summary graph training on various graph summaries[5]. In this experiment we transfer the entity embedding and R-GCN weights obtained with single summary graph training. Between the transferred embedding and the R-GCN model there is no layer or trick that modifies the embedding. We assign each original graph node the entity embedding of the representing summary graph node. The embedding is frozen (no gradient) after transfer. After embedding and R-GCN weight transfer, the R-GCN weights have a gradient. We evaluate the models on validation accuracy during original graph training. On the test set, we report the accuracy, F1 weighted and F1 macro scores.

Single Summary Graph Experiments: Results We observe differences in the accuracy validation curves between the single summary graph training models and the multi summary graph training models. This difference in accuracy validation curve indicates a difference in scaling of R-GCN training. We compare the accuracy validation curve of training with the single In/Out graph summary(Figure19, Appendix C) with the curve of the *summation* model(Figure 5), pre-trained on multiple attribute summaries. The y-axis intersection of the In/Out transfer model is at 0.09. The *summation* model intersects the y-axis at 0.39. Furthermore, we measure that the validation accuracy increase persists slightly longer in early epochs for the summation *model* compared to the validation accuracy curve of the In/Out model (Figure 19). In epoch 10 the *summation* model predicts on average with a 87.56% accuracy on the AIFB validation set. The In/Out summary model predicts on average with an 84.80% accuracy in epoch 10 on the AIFB validation set. The validating accuracy of the In/Out attribute summary model, after 51 training epochs, is significantly higher compared to the *baseline* validation accuracy at epoch 51.

We measure comparable differences between the *summation* and the Out attribute summary model on the AM dataset regarding the intersections with the y-axis and the validation accuracy curve. For MUTAG, the intersection with the y-axis is higher when multiple summary graph are used for summary graph training compared to a single summary graph.

In Table 6 the accuracy, F1 weighted and F1 macro results of the single summary graph experiments are displayed. Table 6 indicates that the attribute summaries overall performed better compared to the (k)-forward bisimulation summaries in terms of accuracy, F1 weighted and F1 macro on the AIFB test set. The

[5] The Out attribute summary and (3)-forward bisimulation summaries were researched in previous work[14]

In/Out attribute summary yields the highest accuracy, F1 weighted and F1 macro scores on the AIFB test set. Summary graph training with the In/Out attribute summary yields a 91.29% accuracy(Table 6) on the AIFB test set, which is comparable to the 92.53% accuracy(Table 5), yield by the *attention* model with the multiple attribute summaries pre-training. It should be mentioned, that the In/Out summary counts more edges than the other attribute summary graphs of the AIFB dataset (Table 10). When taking the compression rate of the summary graphs into account, the In attribute summary is also a remarkable performer on the AIFB dataset. The (3)-forward bisimulation summary graph was the worst performing summary graph (73.77% accuracy) on the AIFB test set (Table 5).

On the MUTAG and AM test set the separate graph summaries yield comparable results to the models where summary graph training on multiple graph summaries occurred. On the AM test set we measured that the Out attribute summary yield the highest accuracy of 99.94%. We measured that the (2)-forward bisimulation yield the highest accuracy of 51.80% on the MUTAG test set. The (k)-forward bisimulation summaries seem to have a better effect on the MUTAG test set performance than the attribute summaries. This result is in line with the results of the *Multiple Summary Graphs Experiments*, where we observed that the (k)-forward summaries outperformed the attribute summaries on the MUTAG dataset (Table 5). We do notice the In/Out attribute summary graph training on the MUTAG test set scores slightly higher on F1 weighted and F1 macro scores. remarkable are the standard deviations for the measured F1 weighted and F1 macro, which are slightly higher for the In/Out summary graph on the MUTAG dataset.

	AIFB			**MUTAG**			**AM**		
Single Summary	**Acc**	$F1_w$	$F1_m$	**Acc**	$F1_w$	$F1_m$	**Acc**	$F1_w$	$F1_m$
In	88.21 ± 0.52	91.74 ± 0.41	59.43 ± 1.08	50.37 ± 0.13	37.78 ± 0.57	4.62 ± 0.18	99.93 ± 0.02	99.93 ± 0.02	71.28 ± 0.07
Out	89.81 ± 0.46	93.83 ± 0.21	60.68 ± 1.16	50.28 ± 0.19	37.95 ± 0.12	4.18 ± 0.06	**99.94 ± 0.05**	**99.94 ± 0.05**	**76.13 ± 0.04**
In/Out	**91.29 ± 0.72**	**94.25 ± 0.37**	**61.32 ± 0.69**	50.97 ± 0.89	**39.28 ± 1.23**	**4.82 ± 0.69**	99.83 ± 0.13	99.13 ± 0.13	76.05 ± 0.05
k=1	88.07 ± 0.48	92.62 ± 0.47	57.51 ± 0.88	51.56 ± 0.60	39.02 ± 0.46	4.08 ± 0.55	99.88 ± 0.06	99.88 ± 0.06	71.32 ± 0.06
k=2	84.21 ± 0.47	90.48 ± 0.41	50.25 ± 1.60	**51.80 ± 0.05**	39.05 ± 0.50	4.04 ± 0.37	99.73 ± 0.09	99.73 ± 0.09	71.29 ± 0.05
k=3	73.77 ± 1.09	84.06 ± 0.56	53.67 ± 1.30	51.50 ± 0.21	37.62 ± 0.15	3.83 ± 0.34	-	-	-

Table 6. Single summary graph training results. The model is initialized by entity embedding and R-GCN weight transfer after single summary graph training. The performance is measured on the test set. The In, Out and In/Out summary belong to the attribute summary set. The k=1, k=2 and k=3 summaries belong to the (k)-forward bisimulation summary set.

7.3 Embedding & R-GCN Weights Transfer Experiments

How do pre-trained entity embeddings and R-GCN weights, obtained with summary graph training, influence scaling of R-GCN training and R-GCN performance on entity type prediction?

With the *Embedding & Weight Transfer Experiments* we evaluate the influence of the embedding and R-GCN weight transfer on R-GCN scaling and R-GCN performance. The attribute and (k)-forward bisimulation summaries are used for summary graph training. After summary graph training, we ether transfer the entity embedding or the R-GCN weights. When we transfer the embedding, we freeze the embedding. When do not transfer the embedding, the node embedding is newly initialized for original graph training and has a gradient. When we do not transfer R-GCN weights, we newly initialize R-GCN weights. In this experiment, the R-GCN weights always contain a gradient. The prediction accuracy on the validation set at epoch 0 is used as a measure for R-GCN scaling. We use the *summation* model only for this experiment. The *attention* and *mlp* model have additional weights besides R-GCN weights. These weights influence the prediction accuracy at epoch 0.

In previous experiments, the F1 weighted and F1 macro scores exposed the class imbalance in the data graphs. We decide not to report on the F1 weighted and F1 macro score for the upcoming experiments to focus on accuracy. We measure the accuracy on the validation and test sets.

Embedding & R-GCN Weights Transfer Experiments: Results The results of the *Embedding & Weights Transfer Experiments* are presented in Table 7. We observe for AIFB, MUTAG and AM, that the prediction accuracy at epoch 0 on the validation sets are better when R-GCN weights are transferred. For example, looking at the AIFB dataset, when (k)-forward bisimulation summaries are used to transfer R-GCN weights only, the *summation* model predicts with 25.17% accuracy in epoch 0. The prediction accuracy is 0.0% at epoch 0 on the AIFB validation set when we do not transfer the R-GCN weights, but only the summed entity embeddings. Similar results are measured for MUTAG and AM, where the initialization of the *summation* model when the R-GCN weights are transferred only seems to be improved, compared to only transferring the entity embedding.

What is remarkable, the highest prediction results on the AIFB and AM test sets are yield, when the entity embedding is pre-trained and transferred. A test set accuracy of 91.01% and 99.90% for AIFB and AM are yield when we only transfer the pre-trained entity embedding. In this case, the R-GCN weights are newly initialized and are updated by backprogation during original graph training.

Results on the MUTAG dataset are contrasting the AIFB and AM results. When the attribute summary graph training is conducted and only R-GCN weights are transferred, the highest accuracy of 70.28% is yield on the MUTAG

test set. When the (k)-forward bisimulation summary graph training is conducted to transfer R-GCN weights only, an accuracy of 69.31% is yield. When the entity embedding is transferred the tests set accuracy is 50.25% for attribute summary graph training and 52.28% for (k)-forward bisimulation summary graph training. These results seem to indicate that the produced entity embedding with summary graph training actually prevents the model from training properly, as we observe similar MUTAG test set accuracy results in the *Multiple Summary Graphs Experiments* (Table 5). When only transferring R-GCN weights, retaining the R-GCN gradient and initializing a new embedding, the model is comparable to the *baseline* model, that trains for more epochs. The validation accuracy of this model on the MUTAG dataset during training epochs is displayed in Figure 18 (Appendix C).

summaries	model	embedding transfer	weight transfer	start acc AIFB	test acc AIFB	start acc MUTAG	test acc MUTAG	start acc AM	test acc AM
-	baseline	False	False	0.0 ± 0.0	72.85 ± 1.05	0.01 ± 0.01	48.67 ± 0.83	0.0 ± 0.0	90.48 ± 0.18
attribute	summation	False	True	17.66 ± 6.05	78.12 ± 1.46	**17.35 ± 4.14**	**70.28 ± 0.33**	**15.80 ± 2.55**	91.34 ± 1.12
(k)-f. bisim.	summation	False	True	**25.17 ± 1.27**	81.49 ± 0.75	15.34 ± 2.46	69.31 ± 0.67	14.04 ± 3.11	91.72 ± 0.11
attribute	summation	True	False	0.0 ± 0.0	**91.01 ± 0.53**	1.30 ± 1.95	50.26 ± 0.20	5.78 ± 7.55	**99.90 ± 0.14**
(k)-f. bisim.	summation	True	False	0.0 ± 0.0	87.04 ± 0.17	0.26 ± 0.19	52.28 ± 0.10	5.12 ± 2.73	99.89 ± 0.07

Table 7. Embedding & Weight transfer results. The *start acc* {*dataset*} is measured on the validation set at epoch 0. The test set evaluation is carried out after original graph training. The results indicate that transferring pre-trained R-GCN weights contribute mostly to a better R-GCN initialization. Highest accuracy on the test set is yield when the embedding is pre-trained and transferred. The MUTAG dataset is an exception where the highest test set result is yield with the R-GCN weight transfer only.

7.4 Freezing Embedding & R-GCN Weights Experiments

How can R-GCN training be scaled and R-GCN performance, on entity type prediction, be maintained or improved, while freezing the gradient of R-GCN weights after pre-training on summary graphs?

We carry out experiments where we evaluate the performance of the *summation*, *mlp* and *attention* models, when R-GCN weights are frozen after summary graph training. Note that, the entity embeddings always remain frozen after transfer in these experiments. The attribute summary set and the (k)-forward bisimulation summary set are used for summary graph training. We also use the single (best performing) summary for summary graph training in combination with the *mlp* and *attention* models. In the *Single Summary Graph Experiments* we measured that the In/Out attribute, (2)-forward bisimulation and the Out attribute summaries yield best predictions on the AIFB, MUTAG and AM test sets, respectively.

After discussing the accuracy on the test sets, we evaluate the validation accuracy curve during original graph training epochs where the R-GCN weights are frozen.

Freezing Embedding & R-GCN Weights Experiments: Results The accuracy results on the test set for the *Freezing Embedding & R-GCN Weights Experiments* are presented in Table 8. The *baseline* accuracy results correspond to the presented results in Table 5. For the *summation* model, as the R-GCN weights are frozen after transfer, we retain the gradient of the embedding. The results of the *summation* model indicate that it is not possible for the *summation* model to scale R-GCN training while the R-GCN weights are not frozen. We measure no prediction improvement during training of the model. The result could be expected as the R-GCN weights account for updating the entity embedding.

In rows four and five of Table 8 we present results for the *mlp* model when the R-GCN weights are frozen after transferring the entity embedding and R-GCN weights. Summary graph training with the attribute summaries in combination with the *mlp* model yields the highest accuracy of 89.50% on the AIFB test set. The standard deviation of the results of the *mlp* model on the AM test set imply fluctuating performance. The performances of the *mlp* models, pre-trained on the attribute and the (k)-forward bisimulation summaries, on the MUTAG and AM test sets are comparable to the *baseline* performance.

Now, the *attention* model combined with summary graph training on (k)-forward bisimulation summary graphs yield a 99.17% AM test set accuracy with a standard deviation of 0.15. This setup of the *attention* model outperforms the *baseline* model, which yields a 90.48% accuracy (Table 8). The validation accuracy curve of the best performing *attention* model on the AM validation set, where the R-GCN weights are frozen after transfer, is displayed in Figure 12. The validation accuracy curve of the *attention* model where we do not freeze the R-GCN grads is displayed in Figure 11. Comparing these curves we notice that the R-GCN training is less scaled, when the R-GCN weights are frozen after transfer (Figure 12). we observe a less steep increase starting around epoch 5. In epoch 20 the *attention* model, where the R-GCN weights are frozen (Figure 12), catches up with the validation accuracy of the model where R-GCN weights are not frozen. The same *attention* model, using the attribute or (k)-forward bisimulation summaries, outperforms the baseline on the AIFB test set. The *attention* model combined with summary graph training on (k)-forward bisimulation summary graphs yields on the MUTAG test set an improved prediction accuracy (51.52%), compared to the *baseline* accuracy (48.67%).

The *mlp* and *attention* model yield remarkable results on the AIFB test set in combination with the In/Out attribute summary graph training (rows 8 and 9 in Table 8). The *mlp* and *attention* model predict with a 87.79% and 86.05% accuracy on the AIFB test set. The *mlp* and *attention* model, combined with the In/Out attribute summary graph, outperform the *baseline* accuracy on the AIFB test set (72.85%) significantly. Using the best performing single summaries, these models yield comparable results to the *baseline* model on the MUTAG and AM test sets. Noticeable, are the relatively high standard deviations on the AM test set, when the R-GCN weights for the *mlp* and *attention* models are frozen, after summary graph training on the Out attribute summary.

summaries	model	embedding transfer	embedding grad.	R-GCN weight transfer	R-GCN weight grad.	test acc AIFB	test acc MUTAG	test acc AM
-	baseline	False	True	False	True	72.85 ± 1.05	48.67 ± 0.83	90.48 ± 0.18
attribute	summation	True	True	True	False	12.04 ± 8.78	33.57 ± 5.96	30.66 ± 4.63
(k)-f. bisim.	summation	True	True	True	False	37.35 ± 2.03	37.0 ± 15.14	36.44 ± 3.90
attribute	mlp	True	False	True	False	**89.70 ± 0.39**	48.73 ± 1.79	89.97 ± 9.67
(k)-f. bisim.	mlp	True	False	True	False	80.88 ± 0.98	50.28 ± 0.56	91.28 ± 4.81
attribute	attention	True	False	True	False	88.74 ± 1.24	49.60 ± 0.71	98.15 ± 2.97
(k)-f. bisim.	attention	True	False	True	False	82.62 ± 1.92	**51.62 ± 0.37**	**99.17 ± 0.15**
single (best)	mlp	True	False	True	False	87.79 ± 1.44	50.40 ± 0.22	89.12 ± 5.14
single (best)	attention	True	False	True	False	86.05 ± 0.78	49.76 ± 0.10	89.48 ± 3.58

Table 8. Results of the embedding and R-GCN weights freezing experiments. The *mlp* and attention model showed to improve R-GCN performance compared to the *baseline* on the AIFB, MUTAG and AM test sets while the embedding and R-GCN weights are frozen.

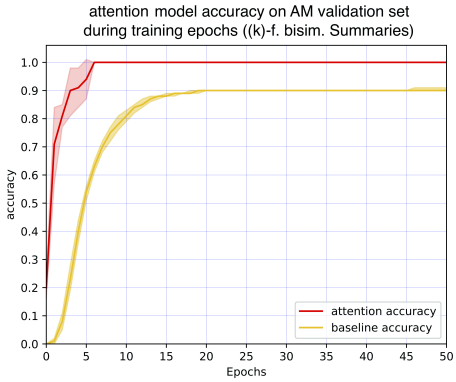

Figure 11. *attention* and *baseline* model accuracy on the AM validation set. The (k)-forward bisimulation summary set for summary graph training was use to initialize the *attention* model. The entity embedding and R-GCN weight transfer with the *attention* model, allow for R-GCN scaling while increasing model performance on the validation set compared to the *baseline*.

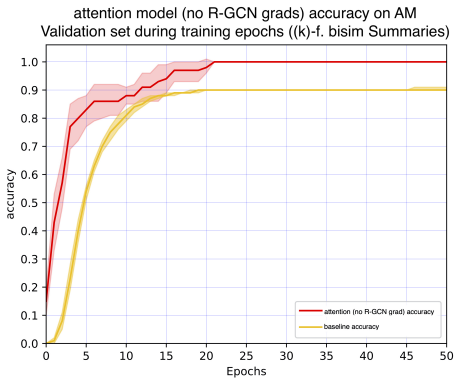

Figure 12. *attention* and *baseline* accuracy on the AM validation set. The (k)-forward bisimulation summary set for summary graph training was use to initialize the *attention* model. The transferred embedding and R-GCN weights are frozen. The attention weights allow for R-GCN scaling while increasing model performance compared to the *baseline*.

Remark 3. In Figures 11, 12, 16 and 17 the upper bound of the standard deviation area of the *attention* models is above 1.0. In Figures 11 and 12 at epoch 5 and epoch 20, respectively, the sample mean plus one standard deviation is higher than 1.0. The following sample example explains how: $sample = 0.95, 0.95, 0.95, 0.95, 0.7$. then, $\mu = 0.9$ and $std = 0.11$. Then it follows $\mu + std > 1.0$. Validation accuracy during training epochs, could never exceed 1.0.

8 Discussion

The results of the *Multiple Summary Graphs Experiments* show that scaling R-GCN training can be achieved with multiple summary graphs, by transferring the multiple entity embeddings adn R-GCN weights with the *summation*, *mlp* and *attention* models. Besides R-GCN scaling, the entity embedding and R-GCN weights transfer improves the test set accuracy significantly for the AIFB and AM dataset compared to the *baseline* model. For the MUTAG the test set results of the transfer models indicate to outperform the *baseline*, but with a less margin. By transferring and freezing an entity embedding trainable parameters are reduced in the transfer models. The *summation*, *mlp* and *attention* model in combination with the attribute summary graph set seem to perform slightly better compared to (k)-forward bisimulation summary graph set on the AIFB dataset. There is no clear difference between the attribute and (k)-forward bisimulation summaries on the AM dataset. For MUTAG it seems that (k)-forward bisimulation summary graph training with the *summation*, *mlp* and *attention* model yield better test set accuracy. The F1 weighted and F1 macro scores of the baseline and transfer models on the MUTAG dataset indicate that there are many classes that are not learned to predict by the models. The F1 weighted and F1 macro scores are caused by the class imbalance in the MUTAG dataset.

There is no clear distinction measured in R-GCN scaling performance between the *summation*, *mlp* and *attention* model in the *Multiple Summary Graph Experiments*. However, the embedding transfer capabilities of the *mlp* and *attention* models come to light in the *Freezing Embedding & R-GCN Weights Experiments*. The results indicate that some setups with the *mlp* and *attention* models are able to scale R-GCN training compared to the *baseline*, while the embedding and R-GCN weights are frozen. The *mlp* and the *attention* layers contain the trainable parameters and are able to modify the entity embedding to improve prediction results for the R-GCN model while the R-GCN weights are frozen. These setups for the *mlp* and the *attention* model outperform or match the test set prediction accuracy on the AIFB, MUTAG and AM datasets. The experiments, most importantly, strongly suggest that the *attention* and *mlp* model could update the R-GCN model without having to retrain R-GCN weights on the original graph. For example, when a new node is added to the graph, new graph summaries can be created. Then, with an existing set of R-GCN weights (frozen) the entity embedding for the summary nodes can be trained. Most likely, the *mlp* and *attention* models are able to transfer the entity embedding to update the R-GCN model while the R-GCN weights remain frozen.

In the *Single Summary Graph Experiments* we measured that pre-training on a single summary graph can yield comparable results to training on multiple summary graphs for the AIFB, MUTAG and AM datasets. Training on a single summary graph, reduces model parameters compared to training on multiple graph summaries. The entity embedding for the original graph is constructed from one summary graph embedding, instead of multiple embeddings. For AIFB, the single summary graphs seemed to perform slightly worse in terms of R-GCN scaling and R-GCN performance compared to the multi-summary graph train-

ing. The In/Out attribute summary graph performed best on the AIFB test set, yielding 91.29% accuracy. The differences in validation accuracy curves of the single summary graph models and the multiple summary graph models can be explained by the amount of training epochs. Each summary graph trains for 51 epochs in summary graph training. This means that in the case of the *summation* model of Figure 5 the summary graph training counts 153 epochs. The In/Out summary model of Figure 19 (Appendix C) used one summary graph, accounting for 51 epochs of summary graph training. More epochs of summary graph training means that the relation specific R-GCN weights are more exposed to the relations in the summary graphs. Furthermore, the *Embedding & R-GCN Weights Transfer Experiments* revealed that for the AIFB and AM datasets, the R-GCN weight transfer is of significant importance for the validation accuracy at epoch 0. It seems that the model with multiple summary graph pre-training accounts for a better model performance at epoch 0. Therefore, it can be logically explained that the validation accuracy at epoch 0 is lower for models that train on one summary graph compared to models that train on multiple summary graphs.

Results of the *Embedding & R-GCN Weights Transfer Experiments* showed that the MUTAG test set prediction improved if only R-GCN weights were transferred. By only transferring R-GCN weights model parameters are increased, as an entity embedding exists for each node in the original graph. Not transferring an entity embedding, increases the number of trainable parameters to be equal to the number of trainable parameters of the *baseline* model. The transfer model, where only pre-trained R-GCN weights are transferred is comparable to the *baseline* model trained for more epochs. The loss curve of the *baseline* model on the MUTAG dataset indicated that the model has not converged after 51 epochs (Figure 8). Increasing training epochs for the baseline model on the MUTAG dataset, most likely improves the test set performance and could potentially outperform the embedding transfer models proposed in this work.

Previous work found that R-GCN training can be scaled with a single graph summary [14]. Furthermore, the research showed that prediction performance of the R-GCN model can improve by transferring R-GCN weights to a new model for training on the original graph. Our results indicate that scaling R-GCN training can be achieved with summary graph training. With multiple or on a single summary graph training an entity embedding and R-GCN weights can be pre-trained and transferred for training on the original graph. Comparing our *baseline* model to the "benchmark" model of previous work[14], we find different accuracy scores on the AIFB, MUTAG and AM test sets after 51 training epochs. The difference in accuracy scores of the *baseline* model can be explained by the fact that the accuracy is differently calculated. We use the accuracy calculation from related work[40], which evaluates each label separately in every prediction. Also, our transfer models exploit different loss and activation functions for the MUTAG and AM dataset compared to previous work[14]. For the MUTAG and AM dataset our models use the cross-entropy loss and the Softmax activation function, as proposed in related work[31, 40].

Related work on graph summarization concluded that attribute summaries are the most viable option for graph summarization of heterogeneous graphs [8]. Our research does not conclude whether the attribute or (k)-forward bisimulation summarization in general have a better influence on scaling R-GCN training and R-GCN performance.

We demonstrated, how the attribute summarization can be integrated in the training pipeline to scale R-GCN training, while improving or maintaining R-GCN performance. Creating graph summaries within the training pipeline scales R-GCN training, reducing parameters and training time. The creation of the (k)-forward bisimulation summaries with FLUID[4], on the other hand, is computationally intensive and relatively time consuming. We want to emphasize that the FLUID framework was not designed for this specific task. It is therefore wrong to undervalue the (k)-forward bisimulation summary creation because of its relatively long computation time. We found that for scaling R-GCN training with graph summaries, we agree with related work[8] that the attribute summaries are a viable option. We did not observe that (k)-forward bisimulation summaries are not a viable option for scaling R-GCN training. (k)-forward bisimulation summaries showed to scale R-GCN training and improve R-GCN performance as well.

The graph summaries created in this research for the MUTAG and AM dataset show high compression rates (Table 10). We encounter graph summaries where the number of summary nodes and edges are less than 1% of the original amount of nodes and edges. For example, the In attribute summary of the AM graph consists of 420 nodes and 7611 edges, while the full AM graph counts 1.6M nodes and 5.9M edges. The AM dataset is constructed from archive data [5]. The archive data could be well structured and could result in a more structured and less heterogeneous data graph. This may explain the relatively small graph summaries for the AM dataset. For the AM graph, the results indicate that R-GCN performance improved, when summary graph training occurred on the highly compressed graph summaries. The compression rated of the MUTAG graph summaries rely on compound/molecule characteristics. In the MUTAG dataset a compound is built from atoms (entity) and bonds (entity). A compound in the MUTAG dataset is often contains the *hasbond* or *hasatom* predicate. The compounds are build with very similar attribute sets. Also, the MUTAG dataset contains 22 relation types (Table 10). This number is relatively small, compared to the size of the MUTAG dataset and the number of relations in the AIFB and AM dataset. This may cause the compression rates of the attribute graph summaries of the MUTAG dataset. On the MUTAG dataset we measured that the test set prediction accuracy increased, when pre-trained R-GCN weights were transferred only. Considering the nature of the MUTAG dataset, the node representations being too uniform could explain the contrasting results when R-GCN was trained with the transferred summary graph embedding. It could be the case, that nodes share the same node embedding, while their types differ.

8.1 Limitations & Future Work

This research has limitations and results should be interpreted accordingly. A limitation of this research is that our results are not comparable to the results of related work[31, 40]. A different classifications task is performed in this research. Furthermore, our *baseline* model is based on the originally proposed implementation[31] of the R-GCN model. Our *baseline* model is a simplified implementation as we use l2 regularization on both (two) R-GCN layers. We do not apply basis- and diagonal-block-decomposition. Also, we kept the hidden layers of the R-GCN constant at 16 hidden units for all datasets. The original R-GCN implementation[31] reduces the R-GCN hidden layer to 10 units when experimenting on the AM dataset. Considering these differences in implementation, the relative difference between the performance of our transfer models and our *baseline* model could be skewed. With the original R-GCN implementation[31], it could be the case that the *baseline* model predictions turn out to be higher. However, the R-GCN layers in the transfer models get equipped with the more sophisticated settings as well and may benefit.

The current research exploits the *rdf:type* relation of the AIFB, MUTAG and AM dataset for entity type prediction. This design decision was made to increase the number of training, validation and test instances compared to the prediction task that related literature performs [31, 40]. Related literature performs the entity type prediction task (besides AIFB, MUTAG and AM) on another dataset called BGS. however, the decision to pursue the *rdf:type* prediction task made the BGS dataset not suitable to use: The BGS dataset contained one entity type with enough support throughout the dataset making the prediction task meaningless. Because we did not perform our experiments on this dataset our results become even less comparable to related work.

Another limitation concerns the comparison of the results of single summary graph training with multiple summary graph training. We conducted multi-summary graph training with the attribute and (k)-forward bisimulation set. These summary sets contain both three attribute summaries. For each summary graph we trained for 51 epochs in summary graph training. This accounts for 153 training epochs. For the single summary graph experiments we conducted summary graph training on one summary graph. This accounts for 51 epochs. Therefore, the comparison between the multi-summary graph transfer models and the single summary graph transfer models is skewed. It would be a stronger comparison, if a 153 epochs single summary graph training was conducted.

For future work, we recommend to overcome the experiment design limitations of our work. Also, we suggest to implement R-GCN scaling on the entity type and link prediction task of related work[31, 40]. The correctness of the entity type prediction task of related work can be argued, because of the number of training/test instances [3]. However, results become comparable to related work if the entity type and link prediction prediction task is conducted as proposed.

We recommend to further investigate, whether the *mlp* and *attention* models are able to update the R-GCN model with the use of graph summaries, without

retraining the R-GCN weights. We propose an experiment where a set of trained R-GCN weights is obtained and not updated by backprogation throughout the experiment. Then, the R-GCN weights are used for summary graph training to obtain an entity embedding, while the R-GCN weights are frozen. The embeddings, resulting from summary graph training are transferred with the *mlp* or *attention* model. Original Graph training is conducted while the multi-layer perceptron and multi-head attention layers contain gradients and the R-GCN weights remain frozen. Accuracy on the test set after training must be compared to a baseline. We expect that this set up can account for a graph update for the entity type prediction task, without the need for retraining the R-GCN weights.

Furthermore, experimentation with larger graphs is needed. We assume, in general, that summarization of larger graphs with more entity and relations types, yield larger graph summaries. As large graphs could exceed available resources on a single device, the graph summaries could exceed the available resources as well. In this case, mini-batch training or mini-batch training in parallel over multiple devices as proposed in related work[20, 35], should be considered.

Other important research could be done on what aspects of graph summaries are important for the R-GCN model. Graph summary features for the creation of a node embedding with graph summary training, could be investigated by researching the attention weights of the *Multi-Head Attention* layer, used in this research. For example, visualizing attention weights could reveal important summary graph features. Knowing which graph summary is suitable for scaling R-GCN training is valuable, as training on a single, suitable summary graph may achieve better R-GCN scaling results and R-GCN performance. The ultimate goal is to be able to explain why a certain graph summary is a good graph summarization and why the graph summarization fits the R-GCN scaling task.

As a last suggestion for future work, we recommend to visualize entity embeddings. t-Distributed Stochastic Neighbor Embedding (t-SNE)[24] can be used to visualize high dimensional embeddings in 2-d scatter plots. Comparing embeddings, resulting from training with summary graphs and with the *baseline* model, could provide information about how the entity embeddings contributed to improvement of the R-GCN performance on the AIFB and AM datasets. Also, it can be researched why the entity embedding, resulting from summary graph training with the MUTAG summary graphs, seems to suppress a proper training of the R-GCN model on the MUTAG graph. We started this research, however we were not able to draw conclusions from it in the current work. We provide visualized entity embeddings, created with t-SNE, in Appendix D. The entity embeddings are the summed summary graph embedding, created by the *summation* model.

9 Related Work: Graph Learning Models

Over the last decade, relation and entity type prediction with the use of graph embeddings have been researched and numerous models have been proposed. Both RESCAL [29] and NTN [39] are compositional models, which use tensor products to capture complex relations between graph nodes. However, computing the tensor product for modeling graph relations requires a high number of parameters, making it a computational expensive process. The Holographic Embedding model (HolE) uses circular correlation of entity embeddings to produce more efficient and scalable compositional vector representations [28]. Circular correlation compresses the tensor product, enabling weight sharing for semantically similar interactions and does not increase the dimensionality of the composite representation. Memory complexity of a *(subject, object)* tuple is linear regarding dimensionality d of the entity representation. Comparing to RESCAL, the composite representation would be d^2 due to the tensor product, making the RESCAL significantly larger in size.

Translation distance models calculate the plausibility of a fact as the distance between subject and object entities regarding the relation between them [45]. The translational models TransE [6] and TransH [44] were two effective graph embedding approaches and produced state-of-the-art prediction results in 2013. TransE embeds entities and relations into the same vector space. The core notion behind TransE is that the relationship between two entities in the graph corresponds to a translation of the relation of the two entities in their embedding. However, since TransE has difficulty representing entities that have 1-to-many, many-to-1 and many-to-many relationships, the TransH model was proposed. TransH allows an entity to have alternative representations when it has multiple relations with various relation types [19, 44, 45]. Inspired by TransE and TransH, TransR[19] was proposed. The TransR model separates entity embeddings from relation embeddings to be exploited in different classification tasks. In their paper[19], the authors showed that TransR outperformed state of the art models like TransE and TransH on the datasets WN18 and FBK15 for link prediction and entity classification. TransR requires more parameters to train compared to TransE and TransH [28], but TransE and TransH are argued to lack expressiveness of the graph embedding [26].

ConvE[34] and ConvKB[27] are Convolution Neural Network based models. These convolution models use 2-D kernels to generate expressive feature map that encodes both the local graph structure and features of nodes. These models are parameter efficient but fail to map interactions between triples, instead these models handle each triple independently. The Graph Convolutional Network (GCN) learns node representation with convolution kernels by accounting for neighborhood nodes. This model is designed to learn on directed and undirected graph containing on relation type. The R-GCN model is proposed as an extension on Graph Convolutional Network (GCN) to learn node representations on large multi-relational graphs [31]. R-GCN extends GCN by accounting for different relations and the directions of edges (incoming and outgoing) connected to a node. R-GCN performance on the link prediction and entity type predic-

tion task alongside parameter settings are investigated in the paper[31] proposing R-GCN. Extending the R-GCN research[31], other literature[40] provides a thorough explanation on the R-GCN model. Furthermore, the literature[40] presents new parameter efficient configurations for the R-GCN model and reproduces the results of the original paper[31] proposing the R-GCN model.

In other work[43], the authors argue that assigning equal weights to the neighborhood entities is a shortcoming of graph convolutional models and propose the Graph Attention Network (GAT). With an attention mechanism, GAT assigns varying weights to edges representing the importance of the neighboring nodes. The Relation Graph Attention network is proposed as addition to GAT for attending multi-relation graph nodes. RGAT preforms similar, or in some cases worse compared to the R-GCN model [7].

10 Conclusion

In this research we proposed models to successfully scale R-GCN training with graph summaries, by transferring an entity embedding and R-GCN weights. The transfer models, proposed in this research, reduce trainable parameters compared to the *baseline* model. The results indicated that R-GCN training can be scaled, while improving model performance on the AIFB and AM datasets. Our models were able to scale R-GCN training for the MUTAG dataset as well. We do expect, however, that the baseline model trained for more epochs, would outperform our proposed embedding and R-GCN weights transfer models.

The *summation* model counts the fewest trainable parameters compared to the *mlp*, *attention* and *baseline* model. The *summation* model showed comparable performance to the *mlp* and *attention* model. However, the *summation* model is not appropriate to transfer an entity embedding when the R-GCN weights are frozen after summary graph training. With the *mlp* and *attention* model we demonstrated how R-GCN training can be scaled and that the test set prediction accuracy of the R-GCN model can be improved, while freezing the node embedding and R-GCN weights after summary graph training. This result suggests that the *mlp* and *attention* model may be able to account for a graph update, such as an added node, without the need for retraining R-GCN weights. For future work, we recommend investigating the ability of the *mlp* and *attention* model to update the R-GCN model, without retraining the R-GCN weights.

Furthermore, investigating single summary graphs for scaling R-GCN training is necessary, since an appropriate graph summary could achieve comparable scaling of R-GCN training and R-GCN performance, compared to summary graph training with multiple summary graphs. However, if uncertainty about a suitable graph summary remains, the *summation*, *mlp* and *attention* models are appropriate models to transfer R-GCN weights and entity embeddings of multiple summary graphs to scale R-GCN training and maintain or improve R-GCN performance.

Acknowledgment

I would like to express my gratitude to Michael Cochez for his supervision and support throughout this research. I thank Till Blume for the opportunity to exploit the FLUID framework[4]. I thank SURF and the Vrije Universiteit Amsterdam for using the Lisa compute cluster.

References

[1] Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF Databases. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.C., Schmidt, R.A. (eds.) Reasoning Web Semantic. Technologies for Information Systems. Lecture Notes in Computer Science, vol. 5689, pp. 158–204. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03754-2_4

[2] Bloehdorn, S., Sure, Y.: Kernel Methods for Mining Instance Data in Ontologies. In: The Semantic Web, pp. 58–71. Springer, Berlin, Heidelberg (2007)

[3] Bloem, P., Wilcke, X., van Berkel, L., de Boer, V.: kgbench: A Collection of Knowledge Graph Datasets for Evaluating Relational and Multimodal Machine Learning. In: Verborgh, R., Hose, K., Paulheim, H., Champin, P.A., Maleshkova, M., Corcho, O., Ristoski, P., Alam, M. (eds.) The Semantic Web. pp. 614–630. Springer International Publishing, Cham (2021)

[4] Blume, T., Richerby, D., Scherp, A.: FLUID: A common model for semantic structural graph summaries based on equivalence relations. Theoretical Computer Science **854**, 136–158 (2021). https://doi.org/https://doi.org/10.1016/j.tcs.2020.12.019

[5] de Boer, V., Wielemaker, J., van Gent, J., Hildebrand, M., Isaac, A., Van Ossenbruggen, J., Schreiber, G.: Supporting Linked Data Production for Cultural Heritage Institutes: The Amsterdam Museum Case Study (2012). https://doi.org/10.1007/978-3-642-30284-8_56

[6] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating Embeddings for Modeling Multi-relational Data. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 26. Curran Associates, Inc. (2013), https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

[7] Busbridge, D., Sherburn, D., Cavallo, P., Hammerla, N.Y.: Relational Graph Attention Networks. arXiv preprint arXiv:1904.05811 (2019). https://doi.org/10.48550/ARXIV.1904.05811

[8] Campinas, S.: Graph Summarisation of Web Data: Data-driven Generation of Structured Representations (2016), http://hdl.handle.net/10379/6495

[9] Chaudhri, V.K., Baru, C., Chittar, N., Dong, X.L., Genesereth, M., Hendler, J., Kalyanpur, A., Lenat, D.B., Sequeda, J., Vrandečić, D., Wang, K.: Knowledge Graphs: Introduction, History and, Perspectives . AI Magazine **43**(1), 17–29 (2022). https://doi.org/10.1002/aaai.12033

[10] Chen, Q., Lim, A., Ong, K.W.: D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. p. 134–144. SIG-

MOD '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/872757.872776

[11] Cilibrasi, R., Vitányi, P.M.: Clustering by Compression. IEEE Transactions on Information theory **51**(4), 1523–1545 (2005)

[12] Daza, D., Cochez, M.: Message Passing Query Embedding. In: ICML Workshop - Graph Representation Learning and Beyond (2020), https://arxiv.org/abs/2002.02406

[13] Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)

[14] Generale, A., Blume, T., Cochez, M.: Scaling R-GCN Training with Graph Summarization. arXiv preprint arXiv:2107.10015 (2022). https://doi.org/10.48550/arXiv.2203.02622

[15] Heist, N., Hertling, S., Ringler, D., Paulheim, H.: Knowledge Graphs on the Web - an Overview. In: Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges (2020). https://doi.org/10.3233/SSW200009

[16] Jia, Z., Lin, S., Gao, M., Zaharia, M., Aiken, A.: Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In: Dhillon, I., Papailiopoulos, D., Sze, V. (eds.) Proceedings of Machine Learning and Systems. vol. 2, pp. 187–198 (2020), https://proceedings.mlsys.org/paper/2020/file/fe9fc289c3ff0af142b6d3bead98a923-Paper.pdf

[17] Leeuwen, M.v., Vreeken, J., Siebes, A.: Compression Picks Item Sets That Matter. In: European Conference on Principles of Data Mining and Knowledge Discovery. pp. 585–592. Springer (2006)

[18] Li, C.T., Lin, S.D.: Egocentric Information Abstraction for Heterogeneous Social Networks. In: 2009 International Conference on Advances in Social Network Analysis and Mining. pp. 255–260. IEEE (2009)

[19] Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning Entity and Relation Embeddings for Knowledge Graph Completion. In: Twenty-ninth AAAI Conference on Artificial Intelligence (2015)

[20] Lin, Z., Li, C., Miao, Y., Liu, Y., Xu, Y.: PaGraph: Scaling GNN Training on Large Graphs via Computation-Aware Caching. In: Proceedings of the 11th ACM Symposium on Cloud Computing. p. 401–415. SoCC '20, ACM, New York, NY, USA (2020). https://doi.org/10.1145/3419111.3421281

[21] Liu, W., Chang, S.F.: Robust multi-class transductive learning with graphs. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 381–388 (2009). https://doi.org/10.1109/CVPR.2009.5206871

[22] Liu, Y., Safavi, T., Dighe, A., Koutra, D.: Graph Summarization Methods and Applications: A Survey. ACM Computing Surveys **51**(3) (2018). https://doi.org/10.1145/3186727

[23] Ma, L., Yang, Z., Miao, Y., Xue, J., Wu, M., Zhou, L., Dai, Y.: NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In:

2019 USENIX Annual Technical Conference (USENIX ATC 19). pp. 443–458. USENIX Association, Renton, WA (2019), https://www.usenix.org/conference/atc19/presentation/ma

[24] van der Maaten, L.: Accelerating t-SNE using Tree-Based Algorithms. The Journal of Machine Learning Research **15**(1), 3221–3245 (2014)

[25] Manola, F., Miller, E., McBride, B.: RDF primer, W3C recommendation (2004), https://www.w3.org/TR/REC-rdf-syntax/

[26] Nathani, D., Chauhan, J., Sharma, C., Kaul, M.: Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs. arXiv preprint arXiv:1906.01195 (2019). https://doi.org/10.48550/arXiv.1906.01195

[27] Nguyen, D.Q., Nguyen, T.D., Nguyen, D.Q., Phung, D.: A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. vol. 2. Association for Computational Linguistics (2018). https://doi.org/10.18653/v1/n18-2053

[28] Nickel, M., Rosasco, L., Poggio, T.: Holographic Embeddings of Knowledge Graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)

[29] Nickel, M., Tresp, V., Kriegel, H.P.: A Three-Way Model for Collective Learning on Multi-relational data. In: Icml (2011)

[30] Paulheim, H.: Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. Semantic Web **8**, 489–508 (2016). https://doi.org/10.3233/sw-160218

[31] Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling Relational Data with Graph Convolutional Networks. In: Gangemi, A., Navigli, R., Vidal, M.E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., Alam, M. (eds.) The Semantic Web. pp. 593–607. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93417-4_38

[32] Serafini, M., Guan, H.: Scalable Graph Neural Network Training: The Case for Sampling. SIGOPS Operating Systems Review **55**(1), 68–76 (2021). https://doi.org/10.1145/3469379.3469387, https://doi-org.vu-nl.idm.oclc.org/10.1145/3469379.3469387

[33] Shah, N., Koutra, D., Zou, T., Gallagher, B., Faloutsos, C.: Timecrunch: Interpretable Dynamic Graph Summarization. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1055–1064 (2015)

[34] Shang, C., Tang, Y., Huang, J., Bi, J., He, X., Zhou, B.: End-to-End Structure-Aware Convolutional Networks for Knowledge Base Completion. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 3060–3067 (2019). https://doi.org/10.1609/aaai.v33i01.33013060

[35] Sheikh, N., Qin, X., Reinwald, B., Lei, C.: Scaling Knowledge Graph Embedding Models for Link Prediction. In: Proceedings of the 2nd European Workshop on Machine Learning and Systems. p. 87–94. EuroMLSys '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3517207.3526974

[36] Shen, Z., Ma, K.L., Eliassi-Rad, T.: Visual Analysis of Large Heterogeneous Social Networks by Semantic and Structural Abstraction. IEEE Transactions on Visualization and Computer Graphics **12**(6), 1427–1439 (2006). https://doi.org/10.1109/TVCG.2006.107

[37] Shi, L., Tong, H., Tang, J., Lin, C.: Flow-Based Influence Graph Visual Summarization. In: 2014 IEEE International Conference on Data Mining. pp. 983–988 (2014). https://doi.org/10.1109/ICDM.2014.128

[38] Smets, K., Vreeken, J.: The odd one out: Identifying and Characterising Anomalies. In: Proceedings of the 2011 SIAM International Conference on Data Mining. pp. 804–815. SIAM (2011)

[39] Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with Neural Tensor Networks for Knowledge Base Completion. Advances in Neural Information Processing Systems **26** (2013)

[40] Thanapalasingam, T., van Berkel, L., Bloem, P., Groth, P.: Relational Graph Convolutional Networks: A Closer Look. arXiv preprint arXiv:2107.10015 (2021). https://doi.org/10.48550/ARXIV.2107.10015

[41] Tiwari, S., Al-Aswadi, F.N., Gaurav, D.: Recent trends in knowledge graphs: theory and practice. Soft Computing **25**(13), 8337–8355 (2021). https://doi.org/10.1007/s00500-021-05756-8

[42] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[43] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph Attention Networks. Stat **1050**, 20 (2017)

[44] Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge Graph Embedding by Translating on Hyperplanes. Proceedings of the AAAI Conference on Artificial Intelligence **28**(1) (2014). https://doi.org/10.1609/aaai.v28i1.8870

[45] Zhang, S., Sun, Z., Zhang, W.: Improve the translational distance models for knowledge graph embedding. Journal of Intelligent Information Systems **55**(3), 445–467 (2020)

[46] Zhou, Y., Cheng, H., Yu, J.X.: Graph Clustering Based on Structural/Attribute Similarities **2**(1), 718–729 (2009). https://doi.org/10.14778/1687627.1687709

Appendix

A Attribute Summaries: Extended Example

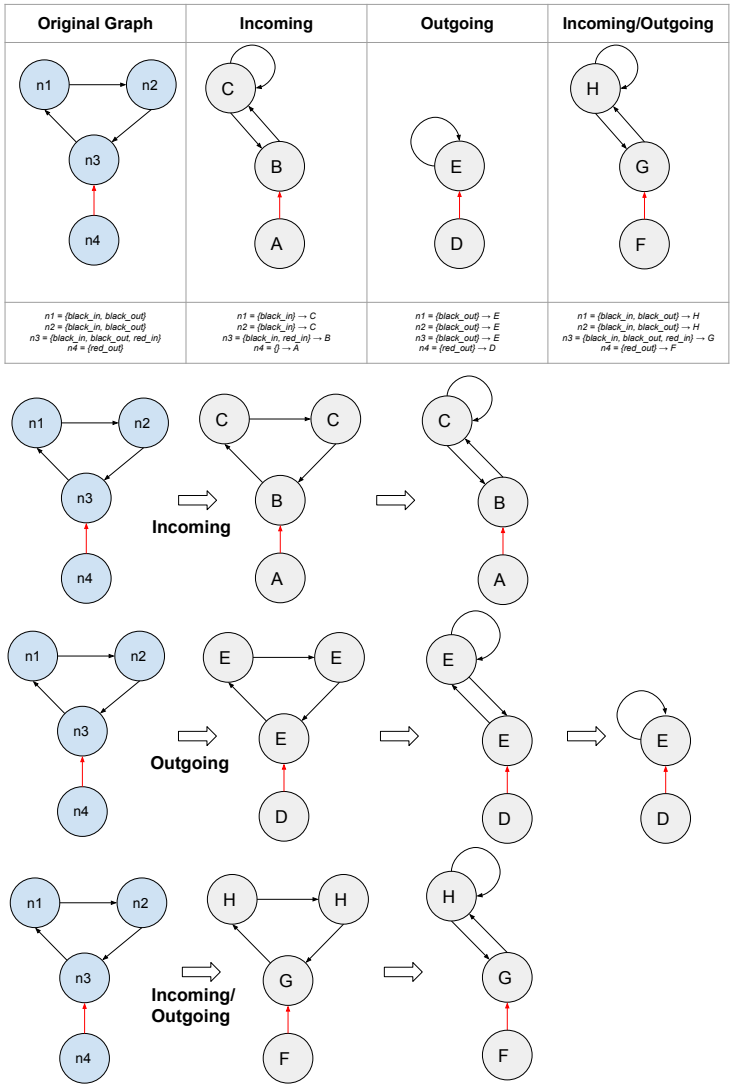

Figure 13. Extended visualization of the creation of the Incoming, Outgoing and Incoming/Outgoing attribute summaries.

B Graph Datasets & Graph Summaries

	AIFB	**MUTAG**	**AM**
Nodes	8,285	23,644	1,666,764
Edges	29,043	74,227	5,988,321
Classes	26	113	21
Relations	45	23	132
Multi labeled	1300	0	3
Single labeled	1024	22534	1028595

Table 9. Graph statistics of the AIFB, MUTAG and AM datasets.

	AIFB		**MUTAG**		**AM**	
Attribute Summaries	Nodes	Edges	Nodes	Edges	Nodes	Edges
Incoming	44	453	11	45	420	7611
Outgoing	359	5518	62	530	6970	195673
Incoming/outgoing	418	7102	70	547	8227	216679
(k)-forward bisimulation						
k=1	363	3291	66	588	7800	148439
k=2	1278	10589	76	620	39463	742917
k=3	2446	19742	85	644	-	-

Table 10. Number of nodes and edges of the summary graphs created for the AIFB, MUTAG and AM datasets.

C Experiment Results

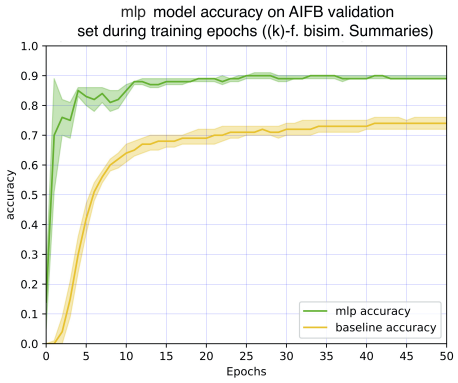

Figure 14. *mlp* and *baseline* accuracy on AIFB validation set during training epochs. The *mlp* model uses embedding and weight transfer from (k)-forward bisimulation summary graph training.

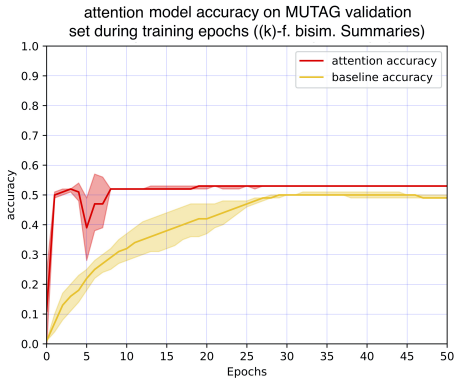

Figure 15. *attention* and *baseline* accuracy on MUTAG validation set during training. The *attention* model uses embedding and weight transfer from (k)-forward bisimulation summary graph training.

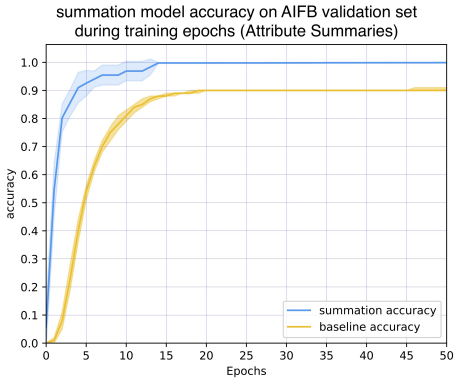

Figure 16. *summation* and *baseline* accuracy on AIFB validation set during training epochs. The *summation* model uses embedding and weight transfer from attribute summary graph training.

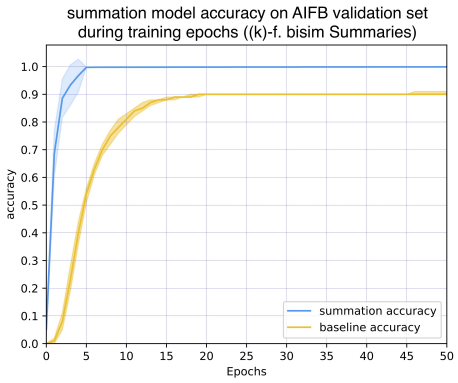

Figure 17. *summation* and *baseline* accuracy on MUTAG validation set during training. The *summation* model conducts (k)-forward bisimulation summary graph training.

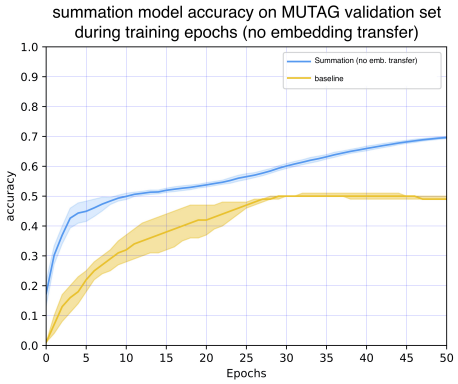

Figure 18. *summation* and *baseline* accuracy on MUTAG validation set during training epochs. The embedding is not transferred to the *summation* model. The R-GCN weights are trained with summary graph training on the attribute summary set. The R-GCN weights contain a gradient after transfer.

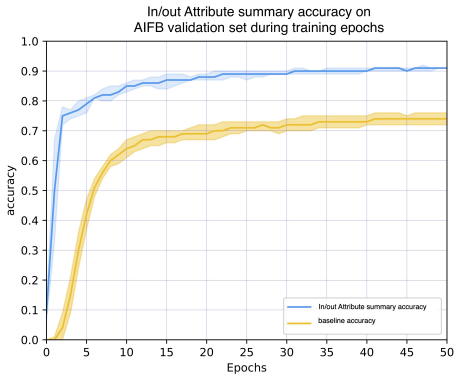

Figure 19. *In/Out* and *baseline* accuracy on AIFB validation set during training epochs. The *In/Out* model conducts summary graph training with the In/Out attribute summary graph. A less-long persisting increase of the validation accuracy curve of the summation model can be observed compared to Figure 5.

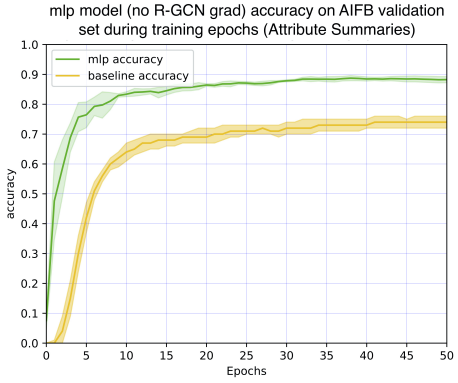

Figure 20. *mlp* and *baseline* accuracy on AIFB validation set during training epochs. The *mlp* model uses embedding and weight transfer from attribute summary graph set training. The embedding and R-GCN weights of the *mlp* model are frozen after transfer.

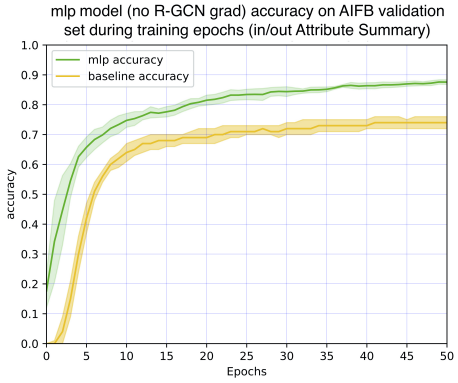

Figure 21. *mlp* and *baseline* accuracy on AIFB validation set during training epochs. The *mlp* model uses embedding and weight transfer from In/Out attribute summary graph training. The embedding and R-GCN weights of the *mlp* model are frozen after transfer.

D Embedding Visualizations with t-SNE

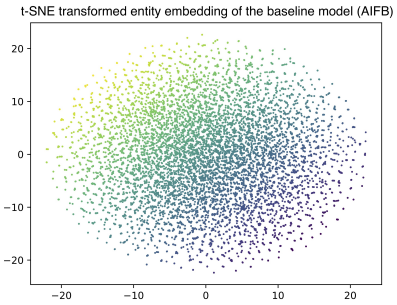

Figure 22. Visualization of the AIFB entity embedding produced with the *baseline* model.

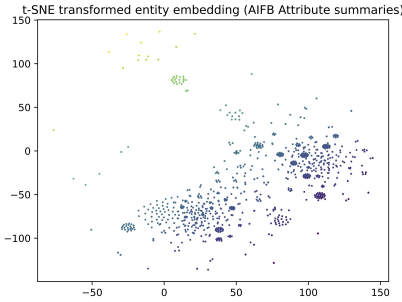

Figure 23. Visualization of the summed entity embedding produced with AIFB attribute summary graphs.

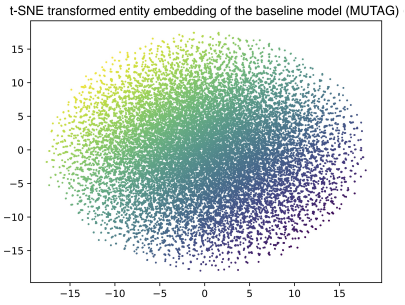

Figure 24. Visualization of the MUTAG entity embedding produced with the baseline model.

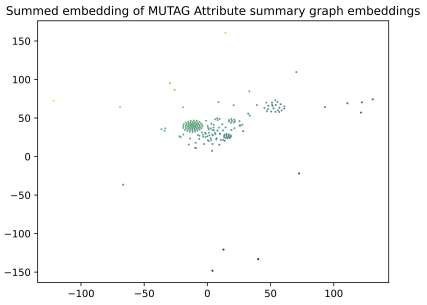

Figure 25. Visualization of the summed entity embedding produced with MUTAG attribute summary graphs.

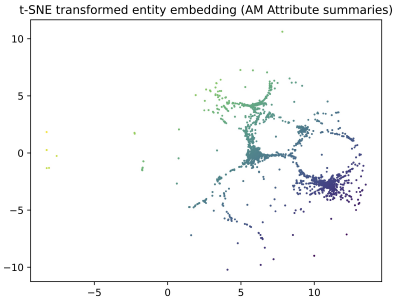

Figure 26. Visualization of the summed entity embedding produced with AM attribute summary graphs.

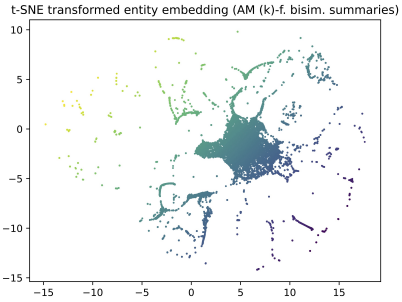

Figure 27. Visualization of the summed entity embedding produced with AM (k)-forward bisimulation summary graphs.