Enabling Representation Learning in Ontology-Driven Conceptual Modeling using Graph Neural Networks

Syed Juned Ali, Giancarlo Guizzardi, Dominik Bork

Enabling Representation Learning in Ontology-Driven Conceptual Modeling using Graph Neural Networks

Syed Juned Ali[1][0000−0002−0710−8052], Giancarlo Guizzardi[2][0000−0002−3452−553X], and Dominik Bork[1][0000−0001−8259−2297]

[1] TU Wien, Business Informatics Group, Vienna, Austria
{syed.juned.ali, dominik.bork}@tuwien.ac.at
[2] University of Twente, Enschede, The Netherlands
g.guizzardi@utwente.nl

Abstract. Conceptual Models (CMs) are essential for information systems engineering since they provide explicit and detailed representations of the subject domains at hand. Ontology-driven conceptual modeling (ODCM) languages provide primitives for articulating these domain notions based on the ontological categories put forth by upper-level (or foundational) ontologies. Many existing CMs have been created using ontologically-neutral languages (e.g., UML, ER). Connecting these models to ontological categories would provide better support for meaning negotiation, semantic interoperability, and complexity management. However, given the sheer size of this legacy base, manual stereotyping is a prohibitive task. This paper addresses this problem by proposing an approach based on Graph Neural Networks towards automating the task of stereotyping UML class diagrams with the meta-classes offered by the ODCM language OntoUML. Since these meta-classes (stereotypes) represent ontological distinctions put forth by a foundational ontology, this task is equivalent to ontological category prediction for these classes. To enable this approach, we propose a strategy for representing CM vector embeddings that preserve the model elements' structure and ontological categorization. Finally, we present an evaluation that shows convincing learning of OntoUML model node embeddings used for OntoUML stereotype prediction.

Keywords: Ontology-Driven Conceptual models · Graph Neural Networks · Representation Learning.

1 Introduction

Conceptual Models (CMs) are essential for information systems engineering in complex domains since they provide explicit and detailed representations of the subject domains. Ontology-driven conceptual modeling languages provide primitives for articulating these domain notions based on the ontological categories put forth by upper-level (or foundational) ontologies. These Ontology-Driven

Conceptual Models (ODCMs) are believed to provide better support for semantic interoperability of the systems as shown in [13] that methodologies and modeling languages based on theoretically principled foundational ontologies can mitigate a number of semantic interoperability problems that arise in concrete application scenarios. The fundamental ontological distinctions embodied in a foundational ontology have been used a conceptual toolbox for supporting ontological analysis, meaning explication and negotiation, and conceptual clarification [2]. ODCM has been further applied for more sophisticated conceptual model modularization mechanism for complexity management [14] and database design [3].

There exist, however, many CMs that have been created using ontologically-neutral languages (e.g., UML, ER). Connecting models created with these languages to ontological categories would bring the aforementioned benefits to them. However, given the sheer size of this legacy base, manually doing this is a prohibitive task. For this reason, an automated approach for suitably addressing this task would significantly advance the state of the art in the field.

Recently, advanced Artificial Intelligence approaches based on deep learning (DL) and Natural Language Processing (NLP) techniques have been used in conceptual modeling to support intelligent modeling assistants [24], model transformation [8], and metamodel classification [35]. However, these approaches limit the CM representation to the CM elements' labels and do not sufficiently exploit the CM structure and real-world semantics to learn the vector representation of these models. For example, Weyssow et al. [35] transform a CM into a tree-based structure where each class has its attributes and associations as children. As a result, they cannot capture the model's graph structure. Konick et al. [20] limit the representation learning of CM to labels. Due to the lack of such knowledge transfer from a CM to its encoding prior to training, the learned ML models do not generalize well to be used as a more robust vector representation of a CM.

To address the problem of predicting the ontological categorization of a CM, we need semantically richer representations of these models. For that, we need encodings that make the semantics of these models and their constituents accessible to the representation learning algorithms. Knowledge Graphs (KG) can effectively organize and represent knowledge to be efficiently utilized in advanced applications by applying different kinds of reasoning (e.g., rule-based and ML-based). KG representation of CMs can comprehensively capture the CM's graph structure and relations between model elements. Therefore, instead of extracting the information from CMs and applying ML algorithms to the extracted data, we can use an intermediary KG representation of models and apply AI techniques to learn their semantically richer representations. In particular, we propose a type of KG that captures the ontological categorizations of the elements constituting an ODCM. This is termed in the following a *Conceptual Knowledge Graph (CKG)*. CKGs embeddings preserve both the model's original structure and its elements' ontological categorization (*Contribution 1*). We then employ these embeddings to enable a Graph Neural Network (GNN)-based approach for automating the task of stereotyping UML CMs (class diagrams) with the meta-classes offered by the ODCM language OntoUML [15] (*Contribution 2*). Since these meta-classes (stereotypes) represent ontological distinctions put forth by the Unified Foundational Ontology (UFO) [15], this task is equivalent to a task of ontological category attribution for these classes.

We employ our representation learning approach to learn the model nodes' vector-based representation (embeddings). These embeddings can assist modelers with intelligent conceptual modeling tasks, like ontology mapping, model auto-completion, and model search. Since OntoUML and UFO are, respectively, among the most used modeling languages and foundational ontologies in ODCM, our proposal makes a clear contribution in advancing the state of art in this field. We present an experimental evaluation to validate our approach and provide a detailed comparative impact analysis of various design choices in our solution architecture. We use the OntoUML FAIR model dataset described in [3].

The remainder of the paper is structured as follows: Section 2 briefly presents relevant background, including a brief discussion about UFO/OntoUML, Knowledge Graphs, (Graph) Representation Learning, and GNNs. Section 3 discusses how our proposals for *i*) CM vector representations (embeddings), and *ii*) ontological categorization prediction advance the state of art in these two enterprises. Section 4 presents the two contributions of this paper, namely, *i*) an approach for transforming OntoUML models to CKGs; and *ii*) an approach for using GNN-based representation learning that employs these CKGs for OntoUML stereotype prediction. Section 5 reports the results of an experimental evaluation of our approach. Section 6 discusses the obtained results and their implications, as well as threats to validity. Section 7 concludes the paper.

2 Background

In the following, we provide a brief background on conceptual modeling, ontologies, knowledge graphs, and graph-based machine learning.

Conceptual modeling is the activity of representing aspects of the physical and social world for communication, learning, and problem-solving among human users [34]. Conceptualizations are entities that are abstractions (of a part) of reality. A modeling language provides a set of modeling primitives that can represent these conceptualizations. CMs represent abstractions using CML primitives, and ontologies define the conceptualizations. An **Ontology** makes the structure of domain conceptualization accessible through an explicit and formal description. **ODCM** extends or supports conceptual modeling techniques by ontological theories that further formalize the conceptual modeling grammars [34], thereby strengthening the ontological commitment of these languages and thus improving the semantic quality of the CML.

Knowledge Graphs (KGs) represent a collection of interlinked descriptions of entities – e.g., objects, events, and concepts. KGs provide a foundation for data integration, fusion, analytics, and sharing [29] based on linked data and semantic metadata. KGs have been recently used for the representation [30,32] of CMs. Such KG-based representations can act as the intermediary representation of CMs to enable ML-based applications on CMs. Existing works (cf. [28]) for creating KGs from structured and semi-structured data exist. Recently, a generic approach has been proposed that is able to transform arbitrary CMs into CKGs called *CM2KG* [30]. However CM2KG focuses only on the element labels and metamodel information. We define the notion of Conceptual Knowledge Graphs (CKGs) as follows: *Conceptual Knowledge Graphs* (CKG) are ontologically enriched KGs representing CMs. KGs are a suitable representation for applying ML and solving question answering, recommendation, and information retrieval. With CKGs, we aim to enable AI-based applications that exploit the full semantic richness of ontologically enriched conceptual models.

Representation learning makes learning algorithms less dependent on manual feature engineering by using DL methods to learn the underlying explanatory factors hidden in the low-level sensory data[4]. In NLP, representation learning is applied to learn natural language (NL) words' representations and then use these representations in various tasks, e.g., sentence sentiment analysis. *Language models* (LM) are trained to learn the vector representations of NL words. Initial works in LMs include GloVe [25], which learns non-contextual word embeddings by learning a global (context-free) embedding for each word. Pre-trained transformer-based LMs such as BERT [10] can learn robust contextual text embeddings and perform better than traditional non-contextual LMs. Moreover, LMs such as BERT apply a masked language modeling approach, where a language model is trained to predict missing words in a text based on the surrounding context. The model is presented with text with some words randomly masked (or hidden) out, and it must generate the missing words based on the remaining words in the sentence.

Graph representation learning (GRL) creates a mapping that represents nodes or entire (sub)graphs as points in a low-dimensional vector space that reflects the original graph's structure, like global positions of nodes in the graph and the structure of local graph neighborhoods [17]. Graph representation techniques for an Encoder-Decoder framework [17] involve two key mapping functions: an *encoder* function, which maps each node to a low-dimensional vector, and a *decoder* function, which decodes the graph information from the learned embeddings. E.g., the decoder might predict an edge between nodes or a node class. GRL optimizes the encoder and decoder mappings to minimize the error between the decoder mapping node embedding from the encoder and the expected statistic value. E.g., if a decoder maps the node embedding to node degree, then GRL minimizes the error between the decoder predicted degree and the node's degree in the initial graph.

The quality of the learned representation, i.e., how well it encodes the intended *meaning* of the data, depends upon the data quality, the architecture of the DL model, and the learning objectives. E.g., different descriptors of a graph element, e.g., label, degree, and node type, will learn node representations to different extents. Different approaches like node2vec [12] and random walk [26] focus on node-level representation learning and graph representations [17].

Graph Neural Networks are neural models that learn graph representations via *message passing* between graph nodes by information aggregation of a node from its neighborhood. In recent years, variants of GNNs such as Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and Graph Recurrent Networks (GRN) have demonstrated good performance on many DL tasks. GraphSAGE [16] generalized the aggregation function (compared to GCN, which uses "mean" as the aggregation function) that generates node embeddings by sampling and aggregating features from a node's local neighborhood. Once the representations are learned, GNNs achieve state-of-the-art performance in link prediction, node classification, graph classification, and graph mining [36].

3 Related Work

Several works have recently proposed ML-based solutions for various conceptual modeling tasks (cf. [6,7]). To use ML, CMs need to be transformed into vectors. In the following, we provide an overview of the existing means to represent CMs in a vector. We thereby mainly focus on encoding the conceptual model characteristics (see Table 1).

Koninck et al. [20] present representation learning techniques for BPMN business processes. They use BPMN model elements' label information and apply doc2vec-like [21] vector representations of activities, traces, and logs and finally aggregate these representations to produce the entire model's vector representations. Luettgen et al. [23] present a similar representation learning technique using word embeddings for business processes' data to improve the encoding. In their work, they add contextual information using the attributes of the activities instead of only the BPMN elements' labels. Both presented works use the learned representations for model discovery, trace clustering, process model selection, and process monitoring.

Burgueno et al. [8] present a Long Short Term Memory Neural Networks (LSTM)-based approach to infer model transformations from UML class diagrams to ER models. Before feeding the models into LSTM, they transform the input CM into a tree-based representation to capture the model elements (e.g., classes, attributes) and their associations. Their approach also captures contextual information and hierarchy, which is not preserved by using only the object's labels and attributes. Similarly, Weyssow et al. [35] present a metamodel concepts recommendation system. They use the data from a dataset of Ecore-based metamodels and train language models over the model elements' data (name and attributes) to learn the word embeddings of the words present in the metamodels. These learned word embeddings are then used to learn the model representations.

Huo et al. [18] propose an approach to detect business process anomalies using graph encodings of process event log data coupled with graph autoencoders. The authors choose GNNs to improve the encoding of the business process's graph structure during representation learning. Berquand et al. [5] present a KG-based approach to enhance data linkage, reusability, and interpretability of engineering

Table 1: Comparison of related works proposing an encoding for CM knowledge

Work	CML	MP	GS	OS
[20]	BPMN	†	✗	✗
[23]	BPMN	✓	✗	✗
[8]	UML, ER	✓	†	✗
[35]	ECore	✓	†	✗
[18]	BPMN	†	✓	✗
Ours	OntoUML	✓	✓	✓

✓Captured ✗Not Captured † Partially captured
MP: meta-properties, GS:graph structure, OS: ontological semantics

models. Furthermore, they augment the KG with a reasoner, an inference engine, and an NLP layer to apply logical reasoning and extract crucial insights.

Table 1 shows that the ontological semantics are not considered in the CM's transformation into a vector-based representation. All presented works focus on using labels and attribute data to create a model's vector representation. Moreover, a tree-based structure only partially captures the structural information of the CMs graph structure and does not capture longer dependencies, i.e., dependencies exceeding each element's direct neighbors, whereas GNNs allow capturing such information to larger depths.

Existing works align or map foundational ontologies with domain ontologies [33]. Felipe et al. [22] propose mapping rules between the noun synsets of Wordnet and the top-level constructs of UFO. Several works use representation learning on ontologies as graphs to achieve ontology mapping [31]. RDF2Vec [27] considers entities' lexical terms for learning node embeddings; however, they treat an entity composed of multiple words as a single entity, which limits generality. OWL2Vec [9] considers word compositions and adds OWL constraints. Junior et al. [19] propose a DL approach that automatically classifies domain entities into top-level concepts using their informal definitions and the word embedding of the terms. However, their work does not consider graph structure contextual information and only relies on textual labels associated with each entity to predict the foundational ontology concept. Graphmatcher [11] is an ontology matching system that uses a graph attention approach to compute a higher-level representation of a class together with its surrounding terms. Regarding representation learning, [11] is similar to our approach, however, their work does not consider ontological semantics from foundational ontologies.

Therefore, in this work, we present a GNN-based representation learning approach for OntoUML models that not only captures the elements' label and attribute information but also considers the CM's ontological semantics from a foundational ontology, CML's meta-properties and the graph structure information of the model.

4 OntoUML Embeddings and Stereotype Prediction

This section presents the two contributions this paper aims to make. In Section 4.1 we present an approach to transform OntoUML models into Conceptual

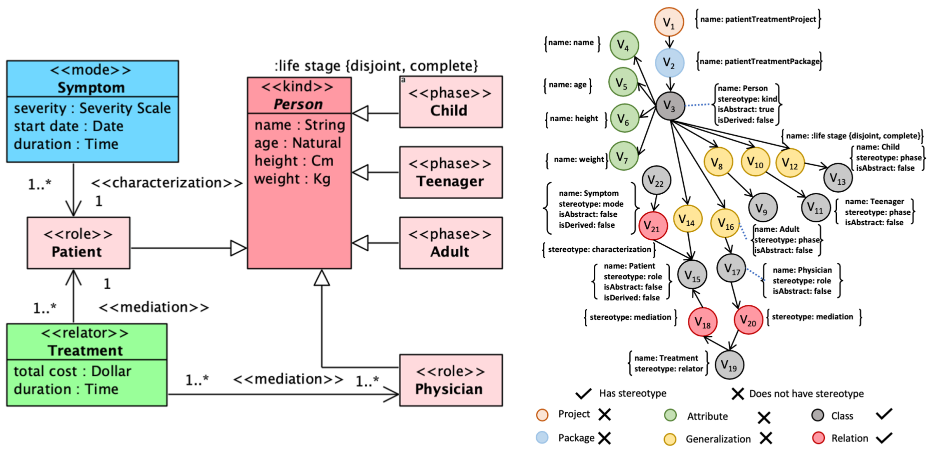
<<mode>>
Symptom
severity : Severity Scale
start date : Date
duration : Time
1..*
<<characterization>>
1
<<role>>
Patient
1
1..*
<<mediation>>
<<relator>>
Treatment
total cost : Dollar
duration : Time
1..*
<<mediation>>
1..*
:life stage {disjoint, complete}
<<kind>>
Person
name : String
age : Natural
height : Cm
weight : Kg
<<phase>>
Child
<<phase>>
Teenager
<<phase>>
Adult
<<role>>
Physician
{name: patientTreatmentProject}
{name: patientTreatmentPackage}
{name: name}
{name: age}
{name: height}
{name: weight}
{name: Person stereotype: kind isAbstract: true isDerived: false}
{name: :life stage {disjoint, complete}}
{name: Child stereotype: phase isAbstract: false}
{name: Teenager stereotype: phase isAbstract: false}
{name: Symptom stereotype: mode isAbstract: false isDerived: false}
{name: Adult stereotype: phase isAbstract: false}
{stereotype: characterization}
{name: Patient stereotype: role isAbstract: false isDerived: false}
{name: Physician stereotype: role isAbstract: false}
{stereotype: mediation}
{stereotype: mediation}
{name: Treatment stereotype: relator}
V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10} V_{11} V_{12} V_{13} V_{14} V_{15} V_{16} V_{17} V_{18} V_{19} V_{20} V_{21} V_{22}
✓ Has stereotype
✗ Does not have stereotype
Project ✗
Package ✗
Attribute ✗
Generalization ✗
Class ✓
Relation ✓

Fig. 1: An example OntoUML model (left) and transformed CKG (right)

Knowledge Graph embeddings. Section 4.2 then discusses how the learned embeddings can be used for OntoUML stereotype prediction.

4.1 Transforming OntoUML models into CKGs embeddings

We will now describe the *OntoUML2CKG* transformation that aims to bridge the CM knowledge encoding gap by incorporating ontological knowledge and graph-structural knowledge in CKG creation – a prerequisite for the GNN-based node embedding learning (see Section 4.2). The transformation is composed of two sequential steps (cf. Figure 2): *i*) *knowledge encoding* by transforming an input OntoUML model into a CKG; and *ii*) transforming the CKG into a vector space to enable GNN-based processing of the CKG.

We will use the example OntoUML model in Fig. 1 to illustrate our approach. The example shows classes and relations with ontological semantics from UFO captured by the stereotype attribute of the classes and relationships, e.g., *kind*, *role*, and *mediation*. In this step, we transform an input OntoUML model into a CKG as shown on the right of Fig. 1. A CKG is a directed graph that consists of nodes and edges. Each node consists of multiple attributes, and each attribute is associated with its value. The attributes capture the *label* information, meta-properties like *isAbstract*, *isDerived*, and the *stereotype* meta-attribute, which captures UFO-based ontological information for an element. Note that there will, of course, be many more meta-properties present; however, we show only *isAbstract* and *isDerived* for simplicity.

Serialized OntoUML models are structured into *projects*, which contain *packages*, and each package has OntoUML *classes*. OntoUML classes are related to each other by *generalization* relationships, which connect abstract to concrete classes and *relation* relationships of different types, namely, *association*, *composition*, and *aggregation*. Each class further contains *attributes*. The stereotype meta-attribute is associated with *class* and *relation*. To transform the input model into a CKG, we create a graph from the model's JSON serialization. During our transformation, we consider the six different structures as nodes as shown in Fig. 1, for e.g., *project* as V_1, *package* as V_2 and *class* V_3. We consider *project* and *package* as a node in our transformation because these structures link multiple models present in a project. We model *generalization* and *relation* relationships also as nodes because this allows us to associate each element in the input model with its attributes. E.g., treating the *mediation* relation between "Treatment" and "Physician" classes as a node allows us to separately add properties to the node as shown for node V_{21} in Fig. 1. Furthermore, this makes our approach suitable for stereotype prediction on the classes and relations. Each relationship is associated with a source and a target class. To model relationships as nodes, we create connections from the source class to the relationship node and from the relationship node to the target class.

Once we have transformed the input model into a CKG, we need to initialize a feature vector for each node in the CKG that will be trained using GNNs to capture the node's semantics. We consider each node's semantics from its NL *label*, its meta-properties, and its associated *stereotype*. Fig. 1 shows a class with the label "Symptom", *stereotype* "mode" and meta-properties associated with this class like "isDerived" and "isAbstract". The GNN algorithm captures the graph structure properties (e.g., node degree) during training of the node embeddings.

The label of each node can contain ontological semantics encoded in NL words. We need a vector representation of a word that captures its contextual information, which can further support predicting the ontological stereotype associated with the label. Therefore we choose BERT as the language model that learns robust contextual word embeddings. However, BERT is a pre-trained model on a large corpus of domain-independent data and, therefore, lacks any OntoUML-specific semantics. To produce domain-specific word embeddings, we therefore also use a GloVe-based language model trained on OntoUML model data exclusively. To that end, we extract all the node and attribute labels data from each OntoUML model in our data set [3], create a data corpus from all the models, and use this corpus to train a GloVe model to learn OntoUML models-specific node label embeddings. To transform the meta-properties into a numerical vector, we create a vector with a set of meta-properties. We then assign a binary value for each meta-property, indicating the presence of that meta-property in a CKG node. We concatenate the two vectors (label and meta-properties, see Fig. 2), and use it as the initial representation of a CKG node. This vector-based representation is still incomplete because it does not yet capture the graph structure and information contributed by the node's neighbors. Once each node in the CKG is associated with a feature vector, each node is also associated with its stereotype label information. We need to train the graphs for a node classification task that predicts the correct OntoUML stereotype for each node. Therefore, we mask (i.e., hide) the stereotype label for 20% of the nodes.

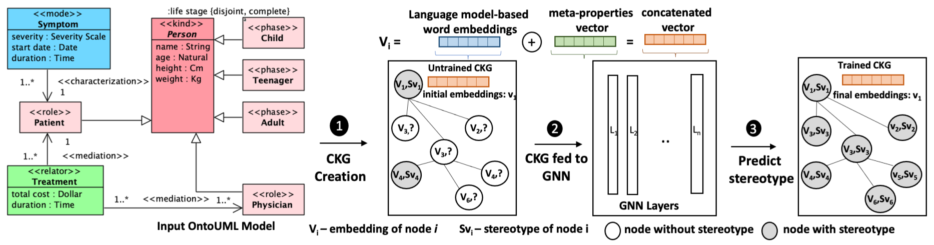

Fig. 2: OntoUML Node Representation Learning Architecture

A transformed CKG with masked labels is shown in Fig. 2 with the nodes as V_i, S_{V_i}. We train the GNN to predict the masked label for each node correctly.

4.2 Using GNN for OntoUML Stereotype Prediction

In the following, we will explain the GNN-based training phase of the CKG nodes' vector embeddings which produces OntoUML node and edge embeddings. Fig. 2 shows all the steps that lead to GNN-based representation learning of OntoUML CMs. Steps 1 and 2, described in Section 4.1, produced GNN suitable representation of the OntoUML CKG.

The CKG is fed to several GNN layers. These layers are responsible for updating or optimizing and thereby "learning" the embeddings. The GNN captures the structure and semantics of the involved nodes, edges, and even entire graphs by aggregating contextual information between neighboring nodes. Fig. 2 shows multiple layers which propagate information in each layer and extract high-level information about the nodes. These layers are usually stacked to obtain better representations [37]. Based on the information aggregation, the GNN model predicts a node's stereotype (class label). The objective function tries to minimize the error in each node between the predicted stereotype label and the true stereotype label. The error is measured by evaluating the binary cross-entropy loss as follows:

$$L = -\frac{1}{N}(\Sigma_{i=1}^{N} y_i.log(\hat{y_i}) + (1-y_i).log(1-\hat{y_i})) \tag{1}$$

where N represents the number of nodes in the CKG while y_i and $\hat{y_i}$ represent the ground truth, and GNN predicted stereotype of the node, respectively. The error is then used to update the node embeddings. Once the training is completed, the final updated embeddings form the learned node embeddings of the CKG, and the stereotype label predicted based on the final learned embeddings forms the final predicted node label as shown in Fig. 2. The results of the training are saved, and test accuracy provides an estimate of the quality of learned node representations. We used a batched graph approach for training the model. However, the model can be trained on each graph with a batch size of 1. Depending upon the batch size, the GNN model can learn model-specific or model-agnostic patterns. We will describe the different configurations in detail in Sect. 5.

5 Experimental Evaluation

A commonly-agreed encoding for ODCM models like OntoUML is missing. In the following experimental evaluation, we account for this research gap by comprehensively discussing and comparing alternative configurations of the GNN architecture and the impact of incorporating specific aspects of the CKG during the training on the model quality. Eventually, we will report on the results of the best-performing configurations when using the trained GNN model for OntoUML stereotype prediction of a partial OntoUML model.

By transforming the models of our OntoUML dataset [3] into CKGs, we produced a CKG dataset composed of 131 models with **32492** nodes and **287259** edges[‡] (cf. Table 2). Each OntoUML class and relationship can have properties connected by edges to the corresponding node. Therefore we see a larger number of edges in Table 2.

Table 2: Dataset statistics

Attribute Nodes (PN)	**FSF**	**# models**	**# nodes**	**# edges**	**# labeled nodes**
With PNs	Unfiltered	131	32492	287259	10400
	FSF 100	101	26472	267199	8982
	FSF 1000	40	12720	200913	3027
Without PNs	Unfiltered	131	19872	250086	10400
	FSF 100	113	18006	243159	9479
	FSF 1000	72	12127	202562	4052

Table 3: Stereotypes with a frequency greater than 100

Stereotype	**Frequency**	**Stereotype**	**Frequency**	**Stereotype**	**Frequency**
subKind	1460	category	412	derivation	184
kind	1155	event	384	participation	170
mediation	1151	roleMixin	337	datatype	126
role	1030	mode	316	type	118
relator	694	characterization	287	collective	113
material	524	phase	267	quality	105
componentOf	436	formal	218	memberOf	101

Each model has a fraction of the total nodes with a stereotype label. After doing a frequency analysis of each stereotype, we selected stereotypes with more than 100 occurrences over all models. We call the stereotypes in this set *frequent stereotypes*. During training, we consider only the nodes associated with a frequent stereotype. We call this node filtering frequent stereotype filtering (FSF). FSF with value N implies selecting nodes with a stereotype with stereotype frequency more than N., E.g., FSF 1000 implies nodes with stereotype frequency more than 1000, e.g., "kind" in Table 3 will be selected. The stereotypes form the different labels that the GNN needs to predict. Note that the connections of nodes without stereotypes with the labeled nodes can still provide crucial semantics. We only mask these nodes, so they are not considered for stereotype

[‡]Note that the edges in Table 2 do not denote the OntoUML relations as they were transformed into CKG nodes to enable their prediction.

prediction. After FSF, we further filter the graphs if less than 20% of the total nodes have a frequent stereotype. We consider FSF 100, which gives 21 different classes, and FSF 1000, which gives four different classes.

In OntoUML models, a stereotype is not defined for nodes of type "Attribute", which reduces the number of nodes with frequent stereotypes. Therefore we distinguish the cases of training the GNN model with and without attribute nodes (AN). Table 2 shows the dataset statistics with a total number of nodes with stereotype labels based on attribute nodes consideration and FSF. Our dataset has around 9000 labeled nodes for classifying nodes in 21 classes and around 4000 for classifying nodes in four classes.

We explained that the textual data is transformed into vectors using two different language models, i.e., GloVe and BERT; therefore, we compare the effect of both LMs on prediction accuracy. Moreover, we also analyze the effect of incorporating the meta-properties on stereotype prediction accuracy by training the GNN, with and without the meta-properties vector. We used a batched training approach that trains the GNN model over batches of CKGs of models. Deep Graph Library [§] provides a way to combine a set of the graph in a batch of fixed size from a set of graphs; therefore, we use two different batch sizes, once with each graph individually and once with all the graphs together (batch size 40, 72, 101, and 131). Note that the different batch sizes result from different configurations, as in Table 2. We use two different variants of GNN models, i.e., Graph Convolution Network (GCN) and GraphSage with max pooling as the aggregation function[†] for GraphSage. GCN uses a weighted sum of the information from all node's neighbors. We train our models with two hidden layers, each of size 128. Therefore, we treat *i*) the GNN model, *ii*) the language model, *iii*) meta-properties, *iv*) attribute nodes, and *v*) batch size as the different configuration parameters. Next, we present the best-performing configurations and then elaborate on the impact of different configuration parameter values.

5.1 Stereotype Prediction Accuracy

We evaluate the prediction accuracy individually using the percentage of correctly predicted stereotypes for each of the discussed configurations.

Table 4 shows the best five configurations with their accuracy values, once with four classes and once with 21 classes corresponding to FSF 1000 (Case 1) and FSF 100 (Case 2). We get a maximum accuracy of 94% with four classes and 67% with 21 classes, and we see that GraphSage outperforms GCN in both cases for all top five configurations.

5.2 Configuration Impact Analysis

Besides the overall accuracy, we are interested to learn how the individual configuration parameters influence the prediction accuracy. Table 5 shows the average

[§]Deep Graph Library: `https://www.dgl.ai/`

[†]This function aggregates the information from a sample of node's neighbors.

Table 4: OntoUML representation learning results

GNN Model	LM	MP	AN	BS	Accuracy
Node classification in four classes (Case 1)					
GraphSage	BERT	Yes	No	1	94.06%
GraphSage	GloVe	Yes	No	1	93.88%
GraphSage	BERT	Yes	Yes	1	93.81%
GraphSage	GloVe	Yes	Yes	1	93.15%
GraphSage	GloVe	Yes	Yes	40	90.20%
Node classification in 21 classes (Case 2)					
GraphSage	GloVe	Yes	Yes	1	67.05%
GraphSage	GloVe	Yes	No	1	61.59%
GraphSage	GloVe	Yes	Yes	101	59.70%
GraphSage	GloVe	Yes	No	113	59.10%
GraphSage	GloVe	No	No	1	44.85%

LM: Language model MP: meta-properties AN: Attribute Nodes BS: Batch Size

Table 5: Configuration parameter impact analysis

Variable	Case 1 Δ Accuracy*	Case 2 Δ Accuracy*
GNN Model	27.45% points ↑ GCN → GraphSage	22.57% points ↑ GCN → GraphSage
Metamodel Properties	21.17% points ↑ No → Yes	18.34% points ↑ No → Yes
Language Model	3.74% points BERT → GloVe	29.91% points ↑ BERT → GloVe
Attribute Nodes	0.49% points ↑ No → Yes	4.80% points ↓ No → Yes
Batch Size	9.78% points ↑ 72 → 1	3.48% points ↑ 113 → 1

change in accuracy of the learned GNN model on the best five configurations, going from one parameter value to another for case 1 and case 2. We find that the choice of the GNN model, the language model, and using meta-properties (in the feature vector) impact the accuracy more than adding attribute nodes. The accuracy increases by 27.45% points for Case 1 and 22.57% points for Case 2, going from GCN to GraphSage. This increase can be attributed to the pooling aggregation of GraphSage, which uses the information only from the most relevant neighbor to learn a node's embedding, thereby reducing the noise in the embedding. Similarly, using the meta-properties also increases the accuracy by more than 18% points in both cases. Meta-properties seem to add more representational semantics to the nodes. Using GloVe over BERT slightly improves the accuracy for Case 1. However, for Case 2, GloVe outperforms BERT by around 29% points. The training of GloVe-based word embeddings using the OntoUML-specific domain semantics seems to impact the accuracy positively. Adding attribute nodes has a poor impact on accuracy, in case 2, almost $-5\% points$. This may be because they add noise during training to the labeled nodes and do not carry enough relevant information to contribute towards the representation of the nodes connected to them. Finally, training the model on individual graphs using a batch size of 1 rather than a single huge graph increases the accuracy. It seems the GNN model is better capable of learning individual model patterns instead of identifying patterns across multiple non-related models. Due to lack of space, we provide details and comparison plots of all configuration parameters in our drive repository[†].

6 Discussion

In the previous section, we presented the performance results of the GNN model for a node stereotype prediction task in OntoUML models. We showed the different possible training configurations and the impact of individual parameters on the prediction accuracy of the GNN model. We trained the GNN GraphSage model to predict the stereotype of an OntoUML model element from four classes and 21 classes with 94% and 67% accuracy, respectively. These results are very promising. The embeddings are learned using message passing in GNN, which implies that the representation of each node reflects its semantics and its relationship with the neighboring nodes. Moreover, the embeddings capture the graph's structural features and stereotype-based ontological semantics, which supports the model in predicting the correct stereotype of the model elements with good accuracy. The stereotype of an OntoUML model element carries rich ontological semantics, and a representation that captures such semantics can be applied in various conceptual modeling tasks where model semantics are crucial. Therefore, the node embeddings learned using the stereotype prediction can now be used for tasks like OntoUML model completion or model search by applying graph-similarity metrics. E.g., cosine similarity between the representation of two graphs can provide an estimate of model similarity, which can be further used to search for similar models or detect model clones. An accuracy of 94% without any hyperparameter tuning of the GNN model shows sufficient potential to be improved using different graph encodings that better capture the ontological semantics and more advanced variants like Graph Attention Transformer.

Of course, this research is not free from threats to validity.

Dataset size – The training dataset for the experiment consists only of 131 OntoUML models. Each model belongs to a specific domain, and the labels consist of domain-specific information, which makes it difficult for the model to learn generalized patterns. We mitigated this by training the OntoUML models over the entire batch of models such that the GNN model learns general patterns and not domain-specific ones. Our batch approach further mitigates this issue as the batch of all the graphs consists of more than 30000 nodes and 10000 labeled nodes.

Labels distribution – Each model consists of nodes with labels; however, certain node types (e.g., Generalization and Attribute) do not have any stereotype, which makes up for about 50% of the nodes. Moreover, the stereotype distribution (see Table 3) shows that the top four stereotypes make up about 21% of the nodes in the dataset. This leads to an uneven distribution of the node classes and affects the classifier model during training. We mitigated this by first training the model on classes with higher frequency, i.e., top four stereotypes, and then

[†]`http://shorturl.at/EHKNT`

training the model on the top 20 classes with at least 100 nodes having that stereotype.

Validation with Pure UML models - We validated our approach on OntoUML models using a fraction of our dataset as a test set. We followed a masked label prediction approach due to which, OntoUML models were suitable for training our GNN models because we could mask a node's stereotype and train the GNN model to predict the masked node, which will not be possible with pure UML models. However, UML models can also be transformed into CKG as our CM to CKG approach is modeling language agnostic and therefore can be used to test on our framework but therefore, testing our approach on pure UML models is part of our future work.

7 Conclusion

In this work, we presented an OntoUML model representation learning approach using an OntoUML to CKG transformation. This transformation encodes the model knowledge, including not only the model elements labels but also UFO-based foundational semantics from the elements' stereotype, meta-properties, the metamodel information, and it preserves the model's graph structure. Our work contributes toward learning semantically richer CMs' embeddings that elevate ML-based conceptual modeling tasks. In our approach, a GNN learns model primitives' vector embeddings trained on a node classification task to classify the stereotype of the CKG node. We achieved this representation learning using an open OntoUML models dataset. We exhaustively explored the dependence on different parameters related to the encoding of knowledge, i.e., meta-properties, attribute nodes, and language model, to understand the impact of these parameters on GNN model training. In the future, we plan to extend our approach towards a larger dataset of the model and use the learned representations on applications like a CMs search [1] where the model similarity is not restricted to the model labels but extends to graph structure information, ontological, and metamodel semantics. Finally, we plan to explore different GNN variants suitable for representation learning of conceptual modeling language primitives.

Acknowledgements

This work has been partially funded through the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 854187.

References

1. Ali, S.J.: Knowledge graph-based conceptual models search. In: Proceedings of the ER Forum and PhD Symposium 2022 (ER 2022). CEUR Workshop Proceedings, vol. 3211 (2022)

2. Amaral, G., Baião, F., Guizzardi, G.: Foundational ontologies, ontology-driven conceptual modeling, and their multiple benefits to data mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **11**(4), e1408 (2021)
3. Barcelos, P.P.F., Sales, T.P., Fumagalli, M., Fonseca, C.M., Sousa, I.V., Romanenko, E., Kritz, J., Guizzardi, G.: A fair model catalog for ontology-driven conceptual modeling research. In: International Conference on Conceptual Modeling. pp. 3–17. Springer (2022)
4. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
5. Berquand, A., Riccardi, A.: From engineering models to knowledge graph: delivering new insights into models. In: 9th International Systems & Concurrent Engineering for Space Applications Conference (SECESA 2020) (2020)
6. Bork, D.: Conceptual modeling and artificial intelligence: Mutual benefits from complementary worlds. CoRR **abs/2110.08637** (2021), `https://arxiv.org/abs/2110.08637`
7. Bork, D., Ali, S.J., Roelens, B.: Conceptual modeling and artificial intelligence: A systematic mapping study. CoRR **abs/2303.06758** (2023). https://doi.org/10.48550/arXiv.2303.06758
8. Burgueño, L., Cabot, J., Gérard, S.: An lstm-based neural network architecture for model transformations. In: ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 294–299 (2019)
9. Chen, J., Hu, P., Jimenez-Ruiz, E., Holter, O.M., Antonyrajah, D., Horrocks, I.: Owl2vec*: Embedding of owl ontologies. Machine Learning **110**(7), 1813–1845 (2021)
10. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint:1810.04805 (2018)
11. Efeoglu, S.: Graphmatcher: A graph representation learning approach for ontology matching (2022)
12. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
13. Guizzardi, G.: The role of foundational ontologies for conceptual modeling and domain ontology representation. In: 2006 7th International Baltic conference on databases and information systems. pp. 17–25. IEEE (2006)
14. Guizzardi, G., Prince Sales, T., Almeida, J.P.A., Poels, G.: Relational contexts and conceptual model clustering. In: IFIP Working Conference on The Practice of Enterprise Modeling. pp. 211–227. Springer (2020)
15. Guizzardi, G., Wagner, G., Almeida, J.P.A., Guizzardi, R.S.: Towards ontological foundations for conceptual modeling: The unified foundational ontology (ufo) story. Appl. Ontology **10**(3-4), 259–271 (2015)
16. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
17. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017)
18. Huo, S., Völzer, H., Reddy, P., Agarwal, P., Isahagian, V., Muthusamy, V.: Graph autoencoders for business process anomaly detection. In: International Conference on Business Process Management. pp. 417–433. Springer (2021)
19. Junior, A.G.L., Carbonera, J.L., Schimidt, D., Abel, M.: Predicting the top-level ontological concepts of domain entities using word embeddings, informal definitions, and deep learning. Expert Systems with Applications **203**, 117291 (2022)

20. Koninck, P.D., Weerdt, J.D., et al.: act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In: International conference on business process management. pp. 305–321. Springer (2018)
21. Lau, J.H., Baldwin, T.: An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv preprint arXiv:1607.05368 (2016)
22. Leão, F., Revoredo, K., Baião, F.: Extending wordnet with ufo foundational ontology. Journal of Web Semantics **57**, 100499 (2019)
23. Luettgen, S., Seeliger, A., Nolle, T., Mühlhäuser, M.: Case2vec: Advances in representation learning for business processes. In: International Conference on Process Mining. pp. 162–174. Springer (2020)
24. Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., Búr, M., Burgueño, L., Engels, G., Jeanjean, P., et al.: Opportunities in intelligent modeling assistance. Software and Systems Modeling **19**(5), 1045–1053 (2020)
25. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
26. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
27. Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: Rdf2vec: Rdf graph embeddings and their applications. Semantic Web **10**(4), 721–752 (2019)
28. Ryen, V., Soylu, A., Roman, D.: Building semantic knowledge graphs from (semi-) structured data: A review. Future Internet **14**(5), 129 (2022)
29. Sequeda, J., Lassila, O.: Designing and building enterprise knowledge graphs. Synthesis Lectures on Data, Semantics, and Knowledge **11**(1), 1–165 (2021)
30. Smajevic, M., Bork, D.: Towards graph-based analysis of enterprise architecture models. In: Int. Conference on Conceptual Modeling. pp. 199–209. Springer (2021)
31. Sousa, G., Lima, R., Trojahn, C.: An eye on representation learning in ontology matching (2022)
32. Sun, S., Meng, F., Chu, D.: A model driven approach to constructing knowledge graph from relational database. In: Journal of Physics: Conference Series. vol. 1584, p. 012073. IOP Publishing (2020)
33. Trojahn, C., Vieira, R., Schmidt, D., Pease, A., Guizzardi, G.: Foundational ontologies meet ontology matching: A survey. Semantic Web **13**(4), 685–704 (2022)
34. Verdonck, M., Gailly, F., Pergl, R., Guizzardi, G., Martins, B., Pastor, O.: Comparing traditional conceptual modeling with ontology-driven conceptual modeling: An empirical study. Information Systems **81**, 92–103 (2019)
35. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Software and Systems Modeling **21**(3), 1071–1089 (2022)
36. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
37. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020)