A GNN-based Recommender System to Assist the Specification of Metamodels and Models

Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, Phuong T. Nguyen
Università degli Studi dell'Aquila, L'Aquila, Italy
{juri.dirocco, davide.diruscio, phuong.nguyen}@univaq.it, claudio.disipio@graduate.univaq.it

***Abstract*—Nowadays, while modeling environments provide users with facilities to specify different kinds of artifacts, e.g., metamodels, models, and transformations, the possibility of learning from previous modeling experiences and being assisted during modeling tasks remains largely unexplored. In this paper, we propose MORGAN, a recommender system based on a graph neural network (GNN) to assist modelers in performing the specification of metamodels and models. The (meta)model being specified, and the training data are encoded in a graph-based format by exploiting natural language processing (NLP) techniques. Afterward, a graph kernel function uses the extracted graphs to provide modelers with relevant recommendations to complete the partially specified (meta)models. We evaluated MORGAN on real-world datasets using various quality metrics, i.e., precision, recall, and F-measure. The experimental results are encouraging and demonstrate the feasibility of our tool to support modelers while specifying metamodels and models.**

***Index Terms*—modeling, metamodeling, recommender systems, graph neural networks**

I. Introduction

With the increasing complexity of software systems, the application of model-driven engineering methods requires the adoption of advanced tools supporting different modeling activities [1], [2], including the specification of metamodels, models and the development of model analysis and management operations. Today's adopted frameworks e.g., those based on Eclipse EMF[1] typically support canonical functionalities, i.e., drag-and-drop, specification of graphical components, auto-completion, without providing context-related recommendations, which may be helpful for modelers to fulfill their designs [3].

Recently, cutting-edge technologies such as neural networks and NLP techniques have attracted considerable attention from the modeling community [4], [5] as they promote the usage of the so-called intelligent modeling assistants (IMAs) [3]. Altogether, this aims to facilitate the completion of models by providing modelers with insightful artifacts, such as attributes or relationships. Nevertheless, there is still the need to support the automation of modeling activities by offering a convenient way to specify metamodels and models, especially for modelers who are less experienced and need more informative suggestions to complete the tasks at hand.

In this work, we propose MORGAN, a MOdeling Recommender system based on a Graph neurAl Network to support the completion of both models and metamodels. To train MORGAN with a large set of models, we mined popular Java software projects stored in the Maven repository,[2] and use the MoDisco reverse engineering tool [6] to extract a model representation for each project. As the first step, MORGAN produces a textual representation of the considered (meta)model by employing tailored model parsers. Then, relevant features are encoded in a graph-based representation that preserves the internal structure of the represented model, i.e., relationships among defined entities. Such an encoding process is model-agnostic since we produce textual files from different formats commonly used to represent (meta)models, i.e., *ecore*, *xmi*. Afterward, the underpinning GNN model computes a graph kernel function used to assess the similarity among nodes by relying on the extracted graphs. MORGAN eventually retrieves the missing structural features that can be used to complete the models under construction. Furthermore, the system can recommend a ranked list of similar classes that modelers can embed to enrich models.

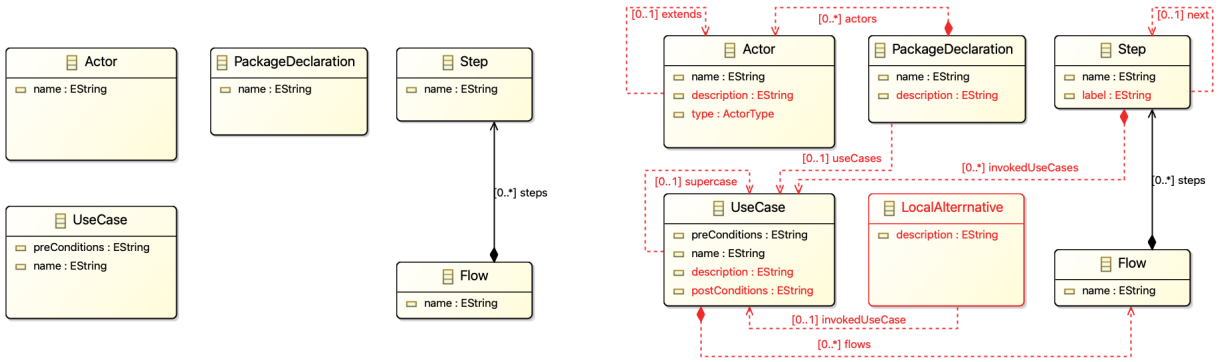
To the best of our knowledge, this is the first approach that exploits GNNs to assist the specification of both metamodels and models. Therefore, we cannot find a comparable tool that can be used as a baseline for the evaluation. We assess the performance of our proposed approach concerning different quality metrics, i.e., success rate, precision, recall, and F-measure. Though we witnessed a reduction in performance for some configurations, where there is a lack of proper training data, overall the obtained results demonstrate that MORGAN can retrieve missing elements needed to complete the metamodels and models under construction. In this respect, the proposed system is capable of providing modelers with a practical means to perform their tasks.

The main contributions of our work are summarized below.

- A framework to assist modelers while specifying both metamodels and models;
- A dataset consisting of models curated from Java projects to enable such kind of modeling support;
- An empirical evaluation on real-world datasets to study the proposed approach;
- A replication package is made available to facilitate future research [7].

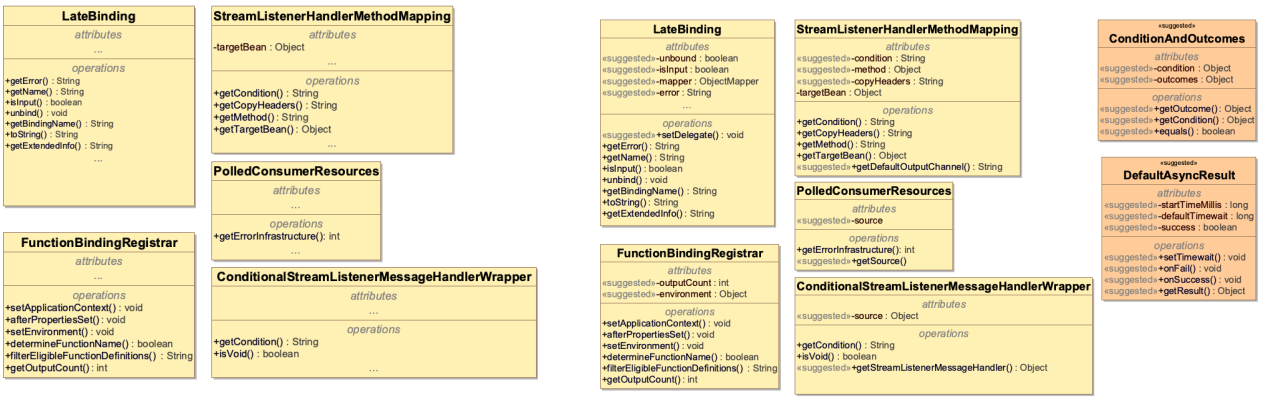
[1]https://www.eclipse.org/modeling/emf/

[2]https://mvnrepository.com/



(a) Partial metamodel (b) Complete metamodel

Fig. 1: The illustrative UMLDSL metamodel.



(a) Partial Java model (b) Complete Java model

Fig. 2: The explanatory SPRINGBOOT model.

The paper is structured as follows. Section II presents an illustrative example to motivate our proposed approach as well as an overview of the GNN algorithm. Section III describes the architecture and main components of MORGAN. The evaluation methodology is presented in Section IV, while the obtained results are discussed in Section V. Afterwards, Section VI reviews relevant existing work in the domain, and finally Section VII concludes the paper.

II. MOTIVATION AND BACKGROUND

Section II-A introduces explanatory examples to show traditional operations involved in the modeling activities. This section also motivates our work by describing two scenarios where automated support is provided to help modelers complete their tasks. Section II-B presents an overview of graph neural networks (GNNs) and graph kernel similarity and how they can be used to cope with the identified issues.

A. Motivating example

We simulate a modeler who is specifying a metamodel/model, and at certain point in time, due to either the complexity of the required task or the lack of experience, she does not know how to proceed. In this respect, a model assistant is expected to help enhance the specification of a partial model by recommending new elements as illustrated below.

▷ **Metamodeling assistant.** Figure 1 represents the explanatory UMLDSL metamodel. At the time of consideration, the modeler has defined only basic attributes and one relation between the *Step* and *Flow* entities. In particular, the initial metamodel specified in Fig. 1(a) defines the key concepts to represent a UML *UseCase*, i.e., *Actor*, *Step*, and *Flow*. Figure 1(b) represents the final version of the corresponding metamodel, where the *PackageDeclaration* entity is completed with *use cases* and *actors* references, and each use case can have several *flows* composed of a sequence of *steps*. By comparing the two sub-figures, we see that some assistances are needed to enrich the partial metamodel shown in Fig.1(a), e.g., by adding new classes, attributes or relations. Moreover, it is also necessary to suggest new metaclasses including structural features, i.e., attributes and references. For instance, as seen in Fig. 1(b), the final *LocalAlternative* metaclass includes the *description* and *includedUseCase* structural features, and this makes the original metaclass more informative/complete. In this way, the assistant should be able to provide the modeler with useful recommendations to help her finalize the metamodeling activities.

In fact, metamodels are used to represent concepts at a high level of abstraction. To design real systems, MDE makes use of models that usually conform to a corresponding metamodel. As a result, recommending additional elements to be incorporated in modeling activities is also meaningful. We show the extent to which the recommendation of models is useful in the following use case.

▷ **Modeling assistant.** In this scenario, the MoDisco framework[3] was exploited to extract models from compilable Eclipse projects. Figure 2(a) represents an excerpt of the Java model of the Spring bootproject[4] extracted by MoDisco. Such a model conforms to the MoDisco Java metamodel covering the full Java language and constructs, i.e., packages, classes, methods, and fields. In this way, existing Java programs can be represented as MoDisco Java models. In particular, Fig. 2(a) depicts a partial Java model with the following five classes:

1) *LateBinding*;
2) *FunctionalBindingRegistrar*;
3) *StreamListenerHandlerMethodMapping*;
4) *PolledComsumerResources*; and
5) *ConditionalStramListenerMessageHandlerWrapper*.

All these classes are incomplete at the time of consideration, and the modeler is supposed to add the missing items. For the sake of presentation, we show the general structure of the Spring boot Java model in terms of classes, methods, and fields. As it can be seen, similar to the above mentioned UMLDSL metamodel, an incomplete Java model can be enriched with new classes, methods, and fields. In Fig. 2(b), we represent a set of possible model elements to complete the partial model. In particular, the model elements tagged by *suggested* are the ones that should be recommended for completing the model. For instance, two classes including their methods and fields are recommended, i.e., *ConditionAndOutcomes* and *DefaultAsyncResult*. Moreover, the model assistant is expected to recommend missing class elements, e.g., the *error* field and the *getError* method for the *LateBinding* class.

The two use cases presented above raise the need for a model assistant to support the completion of partially defined metamodels and models. Since modeling activities are strongly bounded by the domain, a modeler who has limited knowledge of it may encounter some difficulties. Thus, we propose a recommender system that relies on two main concepts, i.e., neural networks applied to graphs and kernel functions, to assist both metamodeling and modeling activities. The following subsections describe these two building blocks in detail.

B. Graph Neural Networks and Kernel Similarity

Graph neural networks (GNNs) have been conceptualized to learn from information structured as graphs [8]. GNNs rely on an information diffusion mechanism in which each node of the graph is processed as a single unit according to the graph connectivity. During the learning phase, the GNN model updates the state of each unit to reach a unique stable equilibrium, i.e., a unique solution given the input.

Formally, for each node *n* the model attaches a state x_n that contains a feature denoted by *n*. Based on this information, the model produces a certain weight as an output namely o_n. Given the local transition function f_w that expresses the dependence of a node on its neighborhood and the local output function g_w that describes how the output is produced, x_n and o_n are defined as follows:

$$x_n = f_w(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}) \tag{1}$$

$$o_n = g_w(x_n, l_n) \tag{2}$$

where l_n is the label of *n*, $l_{co[n]}$ is the edges' labels, $x_{ne[n]}$ are the corresponding states, and $l_{ne[n]}$ are labels of the nodes in the neighborhood of *n*.

GNNs have been applied mostly in molecular chemistry, and biology [9], [10] as proteins are represented through large graphs. However, the applicability of GNNs in other domains has been demonstrated recently [11]–[13], including recommender systems [14], [15]. Besides the original formulation, different variants have been proposed to cope with particular situations as well as to optimize the overall performance. For instance, spectral models have been employed to optimize the propagation of the knowledge represented in the graph using graph convolutional networks (GCNs) [16]. Graph attention network (GAT) has been conceived to enhance the expressiveness of traditional GCN functionality by deploying a multi-head attention module applied to the neighborhood of a node [17]. To compute the internal weights, a GNN model can employ two different methodologies, i.e., vector-space embedding [18] and graph kernel [19]. The former involves the annotation of nodes and edges with vectors that describe a set of features. Thus, the prediction capabilities of the model strongly depend on encoding salient characteristics into numerical vectors. The main limitation is that the vectors must be of a fixed size, which negatively impacts the handling of more complex data. In contrast, the latter works mostly on graph structure by considering three main concepts, i.e., paths, subgraphs, and subtrees. Such techniques support feature embeddings with mutable sizes, which eventually may lead to interesting results in the modeling domain [20]. Formally, a graph kernel is a symmetric positive semidefinite function *k* defined for a pair G_i and G_j such that the following equation is satisfied:

$$k(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle_{\mathcal{H}} \tag{3}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product defined into a Hilbert Space *H*. Roughly speaking, a graph kernel computes the similarity between two graphs following one of the aforementioned strategies. Thus, the proposed GNN model makes use of such a strategy to compute the similarity among the considered models and metamodels, which are encoded as graphs as described in the next section.

[3]https://www.eclipse.org/MoDisco/
[4]https://spring.io/projects/spring-boot

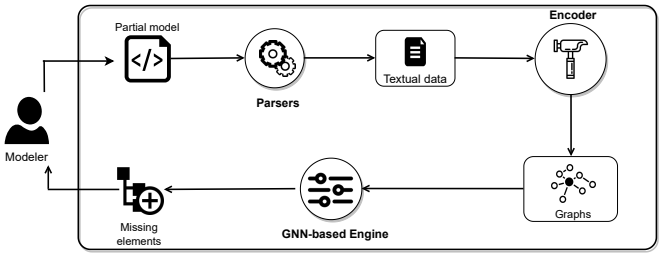


Fig. 3: The MORGAN architecture.

III. The Proposed Approach

In this section, we present in detail the proposed approach, whose architecture is depicted in Fig. 3. Starting from a partial model, MORGAN makes use of tailored model parsers to excerpt relevant information in textual data format. Depending on the nature of the model, we have two independent parsers, i.e., one for metamodels and the other for models. Then, the encoding is performed by the NLP module to obtain graphs used by the underlying *GNN-based engine*. MORGAN eventually recommends the missing elements of the artifact under construction, i.e., either a metamodel or a model.[5] The next subsections give a more comprehensive description for each component.

A. Parsers

This component extracts relevant information from the input artifact by using two different parsers, i.e., *Ecore parser* for metamodels and *MoDisco parser* for models. The former relies on internal Eclipse EMF utilities to parse metamodels as they are expressed in *ecore* format. The latter parses models by interacting with the *xmi* structure.

▷ **Metamodel parsing.** Starting from a metamodel, *Ecore parser* excerpts the list of metaclasses and their structural features, i.e., attributes and references. In particular, the component navigates the tree structure by starting from the root package and its sub-packages to find the mentioned elements. Concerning structural features, each element is represented as a *triple* defined as follows:

- **Name**: The name of the element.
- **Type**: An element is characterized using a certain type. For instance, canonical type extracted from an *ecore* metamodel could be EString or EInt.
- **Relation**: The last component identifies the type of relation between the element and the class, i.e., attribute/reference and method/field for metamodels and models, respectively.

Following the above mentioned scheme, the *Actor* metaclass described in Fig. 1(a) is encoded as follows:

$$Actor\ (name, EString, attribute) \tag{4}$$

where the metaclass has an *attribute* named *name* of the type EString. These elements improve MORGAN's recommendation capabilities as they enable the modeler to add further information to the partial model.

▷ **Model parsing.** *MoDisco parser* is used to extract valuable data from models since *(i)* models express concepts strongly related to Java development, e.g., class declarations, method invocations, interfaces; and *(ii)* MoDisco produces as output a particular *xmi* format that requires internal utilities to be parsed. Thus, each MoDisco model is represented as a list of *Java classes* followed by *method declaration* and *field declaration*. Similar to *Ecore parser*, *MoDisco parser* explores *xmi* tree to elicit valuable elements, i.e., methods and fields that can be used in the recommendation phase. The output conforms to the one produced for the metamodels apart for the type of each element, i.e., the second component of the triple. In this respect, the parser retrieves the *returnType* and the *fieldType* fields depending on the type of the element. For instance, the signature of the `public void write()` method is translated into `(write,void,method)`.

B. Encoder

The next step is to build graphs from textual files produced by the parsing phase. To this end, the artifact *Encoder* component applies a standard NLP pipeline composed of three main steps, i.e., stop-words removal, string cleaning, and stemming. We make use of the implementation of the Porter's Stemmer algorithm provided by the *nltk* Python library[6] to extract the root from each analyzed term. Afterwards, a corpus of words is created from scratch by inserting stemmed model elements iteratively. It is worth noting that a single element is not inserted if it is already included in the dictionary. In such a way, MORGAN encodes relevant information related to the application domain by embedding key features extracted from actual models. Furthermore, this component employs NetworkX,[7] a Python library that creates nodes and edges considering the structure of the parsed model. According to the format shown in Equation 4, each model is represented by a list of connected graphs in which each class is linked with corresponding elements.

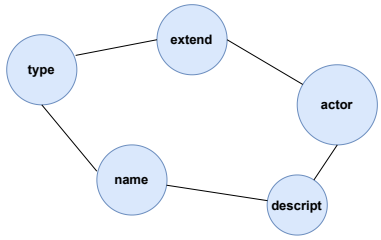


Fig. 4: Example of a stemmed graph.

An explanatory graph obtained by the *Encoder* component is depicted in Fig. 4. It is an encoding fragment of the metaclass *Actor* shown in Fig. 1(b). In particular, the structure of the model is preserved by adding edges among the class *Actor* and the stemmed version of its attributes and references

[5]To facilitate the presentation, the common name "*artifact*" refers to both "*metamodel*" and "*model*," unless otherwise explicitly stated.

[6]https://www.nltk.org/howto/stem.html

[7]https://networkx.org/

namely *Name*, *Description*, *Type*, and *Extends*, though such type of graph does not resemble the semantic relationships occurring in the actual model, i.e., an attribute can refer to a missing metaclass. Nonetheless, analyzing the structure of the model is enough to create the vocabulary that represents the knowledge base of the GNN model.

C. GNN-based engine

At this point, the underpinning *GNN-based engine* can be fed with the computed graphs to retrieve the missing artifacts. To this end, MORGAN relies on the Grakel Python library that provides several graph kernel implementations [21]. We compute graph similarity by comparing the input model with all the elements in the training set. Among possible functions, we opt for the WeisfeilerLehman algorithm [22] as it offers a reasonable computational complexity, i.e., the computation is linear in the worst case. The employed algorithm replaces each vertex with a multiset representation consisting of the original one plus neighbors' features. Given two graphs *G* and *G'* the WeisfeilerLehman kernel is defined as follows:

$$k_{WL}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \ldots + k(G_h, G'_h) \quad (5)$$

where h is the number of iterations, $G_0, G_1, ..G_h$ and $G'_0, G'_1, ..G'_h$ are the encoded sequences of the graph G and G' respectively. Since the employed kernel algorithm is a pairwise operator, MORGAN retrieves an ordered list of classes stored in the training set ranked by similarity scores. The top-5 similar elements are extracted from the whole training set to support two kinds of recommendation: *(i)* specification of missing structural features; and *(ii)* generation of (meta)classes that can be used to enhance the artifact under specification with further concepts.

Being based on this design, MORGAN is able to provide recommendations including both metamodels and models. The succeeding subsection introduces an explanatory example to show to which extent MORGAN is useful for a metamodeling task.

D. Explanatory example

Table I shows the suggested structural features for the UMLDSL metamodel depicted in Fig. 1(a). We consider two different metaclasses extracted from the artifacts under construction, i.e., *Flow* and *Step*. From the ranked list of structural features, we elicit the most relevant ones given the recommendation context, i.e., the bold items in Table I. In particular, the reference *continuation* is the recommended item that the modeler can use to complete the partial metaclass *Step*. Similarly, the metaclass *Flow* could be enhanced with the *finalizeFlow* attribute. Concerning the recommendation of new classes, we consider again the class *Step* as testing. In this case, the retrieved item is the *StepAlternative* metaclass that can enrich the metamodel, even though it is not included in the complete one. We see that MORGAN produces items pertinent to the modeler's context.

TABLE I: Retrieved items for the UMLDSL metamodel.

Context	Recommended item
Step	*attribute* finalState:EString *reference* **continuation:Step** *reference* initialState:initState *reference* finalStates:FinalState
Flow	*attribute* **finalizeFlow:EBoolean** *attribute* eventPatternId:EString *reference* initialState:initState *reference* finalStates:FinalState
Step	*metaclass* **StepAlternative:EClass** *metaclass* Automaton *metaclass* FSM *metaclass* mFSM

Altogether, it is evident that the recommended items are helpful as they are relevant to the given artifact. In this respect, we conclude that for the explanatory example, MORGAN is able to provide the modeler with useful recommendations to complete the given metamodeling activities. In the next sections, we present the experimental methodologies as well as an empirical evaluation of the tool using real-world datasets to study its feasibility in practice.

IV. Evaluation materials and methods

This section describes the research objectives and the experimental configurations used to evaluate MORGAN's performance. In particular, Section IV-A presents the research questions that we address in this paper. The datasets used in the evaluation are described in Section IV-B. Afterwards, Section IV-D presents the metrics employed to evaluate MORGAN's performances. Finally, the validation methodology is detailed in Section IV-C.

A. Research questions

We study the performance of the proposed system by answering the following research questions:

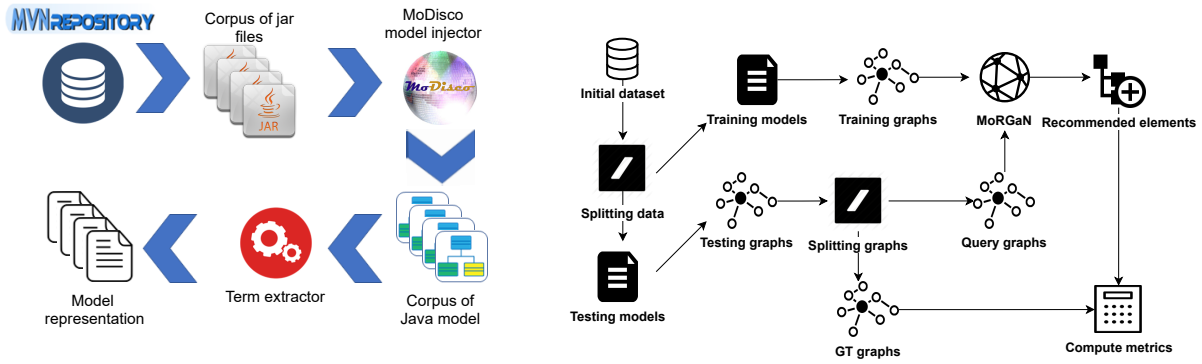
- **RQ**$_1$: *How effective is MORGAN at recommending model elements, including classes and class members?* To assess the prediction performance of MORGAN among MoDisco Java models, we evaluate its ability to provide precise recommendations with respect to classes and fields/methods.
- **RQ**$_2$: *How effective is MORGAN at recommending metamodel elements, including metaclasses and structural features?* We evaluate the performance of MORGAN on recommending metamodel components by investigating different kinds of recommendations, i.e., providing metaclasses, attributes, and references.

B. Datasets extraction

To study the proposed approach, we consider two different datasets, namely D_α and D_β, introduced as follows. Since a large dataset of models is not available, we extracted model representations of popular Java projects stored in the *Maven repository*[8] to build the D_α dataset. The whole process to obtain the required data is depicted in Fig. 5(a). First, we selected the top eight popular categories, including *Apache*, *Build*, *Parser*, *SDK*, *Spring*, *SQL*, *Testing*, and *Web server*, among the most popular ones according to the Maven Tag Cloud.[9] Then, for each category, we crawled around the top 100 popular Java artifacts. The whole process aims to create a balanced dataset composed of good-quality models.

[8]https://mvnrepository.com/

[9]https://mvnrepository.com/tags



(a) The process employed to generate the D_α dataset

(b) The evaluation process

Fig. 5: D_α dataset creation and overall evaluation process.

Figure 6(a) shows the selected ones and the corresponding statistics for the extracted models. The corpus of JAR files has been collected by employing Beautiful Soup,[10] a Python scraping library. Then, Java models have been generated from the collected corpus using MoDisco, an extensible framework that allows us to convert JAR files back to models. Since they are Java models, we extracted three different model elements from the MoDisco models, i.e., classes, methods, and fields. Finally, a model parser is used to represent the model as defined in Section III-A. In the end, we collected a set of 581 unique model representations from the MVN Repository belonging to the top categories.

To populate D_β, starting from an original set consisting of 555 labeled metamodels with nine different categories [23], we extracted metaclasses, attributes, and references from each *ecore* file using the Eclipse EMF utilities. Moreover, aiming to improve MORGAN's performance, we applied different quality filters on the data. In particular, we dropped metaclasses that have less than two elements, either attributes or references. Since each metamodel is encoded as a set of graphs, having small ones may harm the overall performance. Thus, we excluded metamodels belonging to the *Bug* category, which contains only eight metamodels. We also removed possible duplicate classes to avoid any bias. The final dataset consists of eight categories as follows: *Build*, *Conference*, *Office*, *PetriNets*, *Request*, *SQL*, *BibTex*, and *UML*. Figure 6(b) reports a summary related to the characteristics of D_β. Though we do not directly employ the category to produce recommendations, it includes similar metamodels which help to represent the application domain. For both datasets we get rid of small (meta)models and keep only the larger ones since MORGAN is a data-driven approach that heavily relies on the quality of training data.

C. Settings

For the sake of clarity, in the rest of this paper, we refer to the context of metamodeling, i.e., metaclasses, attributes and references. However, it is simple to generalize the same definitions to the modeling activities. To evaluate the prediction performance, we mimic the behaviors of a modeler[11] working at different stages of a modeling project m, by involving different configurations during the experiments [24]. To this end, some parts of m are removed and the rest are used as the modeler's context. In particular, given an original metamodel m, Δ is the total number of the metaclasses. We select δ metaclasses ($\delta < \Delta$) as input for recommendation, while the rest is discarded. This simulates the scenario where the modeler is still working on the project and some metaclasses are missing. In such a way, we simulate different stages of modeling, i.e., an initial model or a mature one. For each metaclass, Π represents the number of original structural features, however only the first π ones ($\pi < \Pi$) are selected as query and the rest is removed and saved as ground-truth data, or *GT(m)*, for future comparison. In practice, δ is small at an early stage, and it increases alongside the development of the metamodel. In the same manner, π is small when the developer starts working on the metamodel. The two parameters δ, π are varied to stimulate different development phases. In particular, we consider the four configurations defined in Table II.

TABLE II: Experimental configurations.

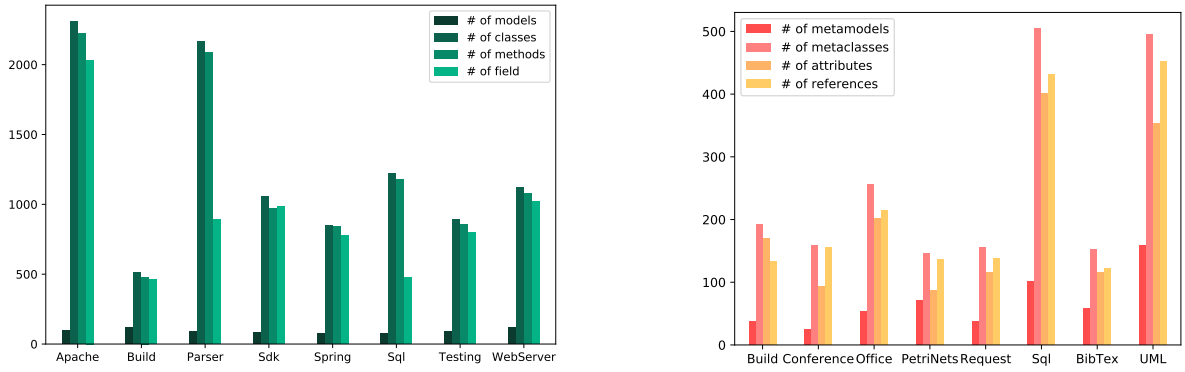
δ \ π	$\Delta/3$	$2\Delta/3$
$\Pi/3$	C1.1	C1.2
$2\Pi/3$	C2.1	C2.2

For instance, the C1.1 configuration ($\delta = \Delta/3, \pi = \Pi/3$) represents an initial stage of the modeling activity when the modeler already defined one third of the classes and structural features. We used these parts as a query and the rest is used as ground-truth data. Differently, C2.2, i.e., $\delta = 2\Delta/3, \pi = 2\Pi/3$, corresponds to the stage where the metamodel under construction m is almost complete. In particular, two thirds of the classes are used as query while the rest is stored as *GT(m)*.

To assess MORGAN's performance, we adopted the k-fold cross-validation technique that is widely used in evaluating

[10]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[11]For the sake of presentation, the two terms "*modeler*" and "*developer*" are used interchangeably in the scope of this paper.



(a) D_α features

(b) D_β features

Fig. 6: Feature of the examined datasets.

ML-based applications [25]. The overall process is described in Fig. 5(b) and it is applied on both the metamodel dataset and the model one presented in Section IV-B. Given the initial datasets, the splitting data operation is performed to obtain training and testing sets. In practice, the former represents the models that have been collected a priori to build the vocabulary of the GNN (see Section III-B, while the latter has been split into *GT graphs* and *query graphs* according to the configurations defined in Table II to mimic a real development scheme. Afterwards, MORGAN is fed with the training graphs to retrieve the missing elements including structural features and metaclasses.

D. Metrics

We consider a set of $|M|$ testing metamodels, and given a partial metamodel *m*, the outcome of MORGAN is a ranked list of N metamodel elements, i.e., $REC_N(m)$. We evaluate the system's performance by comparing such a list with the ground-truth data *GT(m)*. First, we call $match(m) = |GT(m) \cap REC_N(m)|$ as the number of correctly retrieved artifacts, then the following three metrics are used to study the performance: *Success rate*, *Precision*, *Recall*, and *F-measure* metrics, defined as follows [24]:

▷ **Success rate** measures the ratio of testing metamodels that get at least a match in the recommendation list, to the total number of testing metamodels:

$$success\ rate = \frac{count_{m \in M}(|match(m)| > 0)}{|M|} \tag{6}$$

▷ **Precision** is the ratio of number of matched elements to the number of recommended items for a metamodel:

$$precision = \frac{match(m)}{N} \tag{7}$$

▷ **Recall** is the ratio of the ground-truth metamodels being found in the *top-N* recommended items:

$$recall = \frac{match(m)}{GT(m)} \tag{8}$$

▷ **F-measure** is computed as the harmonic average of precision and recall:

$$F - measure = \frac{2 * precison * recall}{precison + recall} \tag{9}$$

V. Results and Analyses

To answer the identified research questions, in Section V-A we report and analyze the results obtained by using the methodology and metrics presented in Section IV. Finally, Section V-B presents the threats that might affect the validity of our findings.

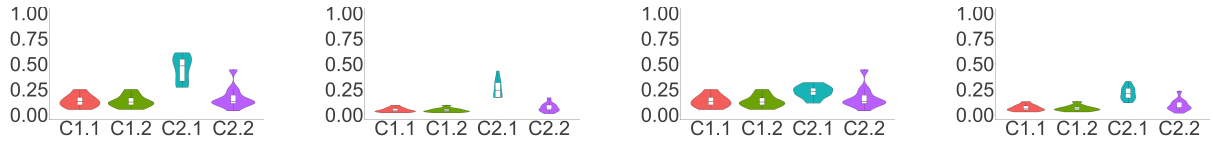
A. Results

To answer the two research questions, we experimented with all the configurations presented in Table II. The experimental results are depicted using violin boxplots, which report both boxplot and density traces. In this way, the plots bring a more informative indication of the distribution's shape [26], allowing us to understand better the magnitude of the density.

RQ$_1$: *How effective is MORGAN at recommending model elements, including classes and class members?* We study the performance of MORGAN in assisting modelers in modeling activities, i.e., the second scenario in Section II. We investigate if MORGAN is able to provide them with relevant model classes, methods and fields as follows.

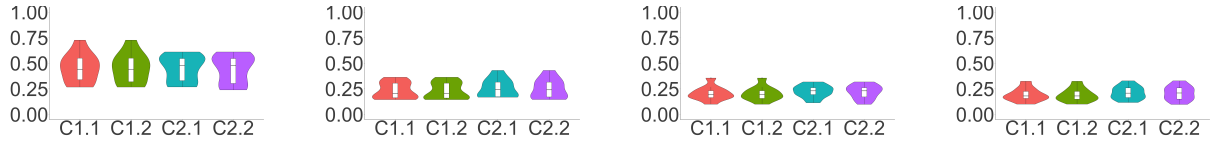
▷ **Recommendation of classes.** The performance of MORGAN in this task is reported in Fig. 7. By all the experimental configurations, we see that the tool can provide relevant recommendations. For instance, the best *success rate* is obtained by C1.1, i.e., ranging from 0.25 to 0.50. Nevertheless, it suffers a low prediction performance with respect to other metrics, i.e., *precision*, *recall*, and *F-measure* as shown in Fig. 7(b), Fig. 7(c), and Fig. 7(d), respectively. For instance, the *precision* scores for C1.1, C1.2, and C2.2 are always lower than 0.25. The same trend can also be seen with the *recall* and *F-measure* metrics.

▷ **Recommendation of class members.** We examine the capability of MORGAN to suggest fields and methods within a model class by referring to Fig. 8. Compared to the results



(a) Success rate (b) Precision (c) Recall (d) F-Measure

Fig. 7: Evaluation scores for recommending model classes.



(a) Success rate (b) Precision (c) Recall (d) F-Measure

Fig. 8: Evaluation scores for recommending model class members.

in Fig. 7, it is evident that MORGAN obtains a better prediction performance for recommending class members. Take as an example, the obtained *success rate* ranges from 0.25 to 0.75 by most configurations. Also by the *precision* metric, the tool substantially improves in relation to the results for recommending model classes. This is further confirmed with *recall* and *F-measure* in Fig. 8(c), and Fig. 8(d), respectively.

Overall, the results in Fig. 7 and Fig. 8 suggest that MORGAN obtains a considerably low prediction accuracy on the D_α dataset. We make an attempt to find out the rational behind such a performance as follows. As MORGAN is a data-driven approach, we attribute a change in performance to the quality of the training data. To validate the hypothesis, we investigate the model dataset, i.e., D_α, by using the EMFCompare tool [27], which has been used to examine models from the structural point of view. Given a list of models belonging to a certain category, we compute the similarity between every possible pair of models to see how they correlate. Table III reports the minimum, maximum and mean values of the similarity obtained for all the eight categories. A similarity score of 1 means that two models are completely identical, while a value 0 corresponds to no similarity.

The table shows an evident outcome: there is a very low correlation among the models within a same categories, i.e., there are not many overlapping projects. For instance, the maximum value is 0.244 obtained by the *Webserver* category, while the minimum similarity is 0.007 for *Sdk*. This means that even with two models in the same category, their extracted features could be very different in terms of classes, methods, and variable type usages. As such, the prediction capabilities of MORGAN are negatively affected as it relies on the concept of commonality among testing and training data.

We suppose that a curated dataset with additional preprocessing steps can help MORGAN improve its performance. To confirm the hypothesis, Table IV shows that metamodels belonging to the same category are more correlated, i.e., they reach higher degree of similarity on average

Answer to RQ$_1$. While MORGAN can provide relevant recommendations with respect to model classes and class members, its prediction performance is heavily driven by the quality of the training data.

RQ$_2$: *How effective is MORGAN at recommending metamodel elements, including metaclasses and structural features?* We analyze the ability or the conceived approach to support modelers by providing them with relevant metaclasses as well as structural features that may help them finalize a metamodel under construction.

▷ **Recommendation of metaclasses.** The evaluation scores obtained by MORGAN for recommending classes are shown in Fig. 9. Concerning *success rate*, as seen in Fig. 9(a), by all the configurations MORGAN obtains a score larger than 0.50 for almost all the testing metamodels. In particular, while the violins span from 0.20 to 0.80, their larger portion is located in the upper part of the diagram, starting from 0.60 to 0.80. This demonstrates that most of the testing metamodels get at least a match for classes from the ranked list of recommendations. Moreover, it is evident that MORGAN obtains a comparable performance by all the four configurations, i.e., the boxplots have a similar shape. With respect to *precision* shown in Fig. 9(b), we see that the distribution of the scores ranges from below 0.10 to above 0.75. However, different from the ones for representing success rate, the violins for precision are more equally distributed, implying that the ratio of correctly retrieved metaclasses to the total number of recommended items varies steadily among the metamodels. By the *recall* metric in Fig. 9(c), we witness a same distribution with that of the success rate violins in Fig. 9(a). Finally, the overall performance of MORGAN is summarized by the *F-measure* boxplots in Fig. 9(d), and it confirms the results exhibited by the *success rate* metric, i.e., the tool is able to provide a match class for most of the testing metamodels.

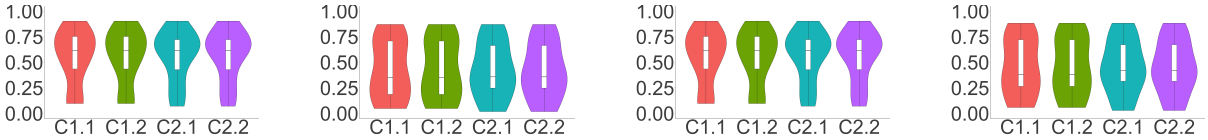
▷ **Recommendation of structural features.** Structural features consist of attributes and references, which are the main components of a class within a metamodel. As shown in

TABLE III: Similarity among models in the D_α dataset.

Category	Min	Max	Mean
Apache	0.052	0.205	0.118
Build	0.022	0.166	0.009
Parser	0.007	0.143	0.080
Sdk	0.007	0.130	0.079
Spring	0.028	0.194	0.009
Sql	0.011	0.213	0.113
Testing	0.007	0.185	0.118
Webserver	0.010	**0.244**	0.093

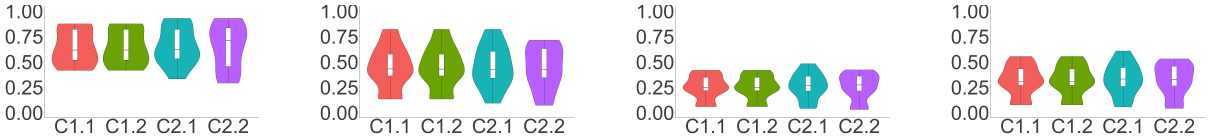
TABLE IV: Similarity among metamodels in the D_β dataset.

Category	Min	Max	Mean
Build	0.000	0.989	0.102
Conference	0.003	0.960	0.215
Office	0.000	0.988	0.047
Petri	0.000	0.993	0.116
Request	0.000	0.953	0.030
Sql	0.000	0.991	0.035
BibTex	0.000	**0.997**	0.106
UML	0.000	0.994	0.086



(a) Success rate (b) Precision (c) Recall (d) F-Measure

Fig. 9: Evaluation scores for recommending metamodel classes.



(a) Success rate (b) Precision (c) Recall (d) F-Measure

Fig. 10: Evaluation scores for recommending metamodel structural features.

Fig. 10, MORGAN obtains a comparable accuracy for all the configurations, i.e., C1.1, C1.2, C2.1, and C2.2. Moreover, the average *success rate* is larger than 0.60 for all the considered configurations (cf. Fig. 10(a)). It means MORGAN is capable of suggesting at least one correct structural feature regardless of the size of the input metamodel. Nevertheless, as seen in Fig. 10(b), Fig. 10(c) and Fig. 10(d), while MORGAN achieves good *precision*, it suffers a decrease in performance with respect to *recall* and thus, *F-measure*. This means that on the given dataset, the tool experiences a low *sensitivity* [28]. It is worth noting that by C2.2, MORGAN achieves better results compared to the other configurations. This is understandable as augmenting the recommendation context helps MORGAN approach more similar training metamodels, allowing it to get more relevant structural features.

Compared to the results in $\mathbf{RQ}_1$, MORGAN yields a superior performance on D_β due to two reasons: *(i)* each category includes metamodels that represent similar meta concepts; and *(ii)* the dataset has been populated and manually classified by expert modelers. As seen in Table IV, the similarity among the considered metamodels is substantially higher compared to that of the models shown in Table III. We conclude that running the tool on a dataset curated with fine tuning techniques brings more relevant recommendations.

Answer to $\mathbf{RQ}_2$. On the given dataset, MORGAN is able to suggest relevant metaclasses and structural features. The tool is more effective in recommending metaclasses, even if the dataset is considerably small.

B. Threats to validity

We discuss the threats that could impact on the validity of the study's outcomes, and identify possible countermeasures to mitigate them. Threats to the *internal validity* involves two aspects, i.e., the chosen GNN model and the employed encoding scheme. Concerning the former, we employ a technique that relies on graph kernels to compute similarity values. Such a comparison can be time-consuming in the case of large graphs. We cope with this issue by using a kernel similarity function that is linear in the worst case. With respect to the latter, some relationships might be missed due to the proposed encoding mechanism. To minimize the threat, we embed all relevant features extracted from textual models.

The selection of the data for evaluation might hamper the *external validity* of our findings. To reduce any potential bias, for D_α we crawled Java software projects from the Maven platform by mining popular projects according to Maven popular tags that identify well-maintained Java projects. Moreover, we used a curated corpus of categorized metamodels as the second dataset.

VI. Related work

This section reviews relevant studies that are related to *(i)* supporting modeling activities; and *(ii)* exploiting GNNs in the recommender systems domain.

A. Modeling assistant tools

The Extremo Eclipse plugin [29] supports modeling activities by analyzing information sources obtained from different

resources, i.e., Ecore, XMI, RDF, OWL files. Extremo uses such excerpted data to build a common data model by mapping relevant entities. The underpinning query mechanism allows users to customize and refine the retrieved entities by following two styles, i.e., predicate-based or custom. The Papyrus plugin [30] was developed to assist modelers in designing domain-specific models. By exploiting the EMF generator model, the approach maps meta-classes conforming to a UML profile in a real Java class. The end-user can even use a graphical context menu to edit the retrieved meta-classes.

Batot and Sahraoui [31] formulate the design of a modeling assistant as a multi-objective optimization problem (MOOP). The system employs the well-known NSGA-II algorithm to search for partial metamodels given a predefined initial set by using the Pareto concept. It helps modelers to complete the delivered models using coverage degrees and pre-defined minimality criteria. Besides textual format specifications, models can be outlined by means of graphical tools, e.g., DIA or Visio. Relying on models fragment in such a format, an example-driven tool was proposed [32] to automatically build a generic metamodel composed of untyped elements. Similarly, Kuschke *et al.* [33] present a pattern-based approach to recommend relevant completions for structural UML models. Each user operation triggers an event in which the detailed information is detected. The tool retrieves a set of ranked activity candidates (AC) which supports modelers during model editing.

Recently, an NLP-based architecture for the autocompletion of partial models has been presented [4]. Given a set of textual documents related to the initial model, relevant terms are extracted to train a contextual model using the basic NLP pipeline, i.e., tokenization, splitting, and stop-word removal. Afterward, the system is fed with the model under construction to automatically slice it following a set of predefined patterns used to search for recommendations. Modelers can give feedback which can be used to update the model and improve the recommendations. A pre-trained neural network was used to recommend relevant metamodel elements, i.e., classes, attributes, and associations [5]. Such data is encoded in tree-based structures that are masked using the well-founded RoBERTa model language architecture plus an NLP pipeline. This preprocessing is needed to obtain an obfuscated training set that are used by the network to produce the outcomes. DoMoBOT [34] combines NLP and a supervised ML model to automatically retrieves domain models from the textual content using the spaCy tool. After the preprocessing phase, the predictive component uses the encoded sentences to retrieve similar model entities and generate the final domain model.

Compared with the reviewed studies, MORGAN is different as it can support various types of models expressed in the supported formats, i.e., *ecore*, *xmi*. Moreover, it is shown that using even partial models in the training phase MORGAN can lead to acceptable results.

B. GNN for supporting recommender systems

GNNs have been successfully exploited to develop several types of recommender systems. Xian *et al.* [35] integrate a repeat-explore mechanism in a GNN called ReGNN to improve the performance of session-based recommender systems. By encoding users' behavior in the graph, ReGNN can recognize user patterns and refine the recommended items. Similarly, a GNN model has been used to integrate an external knowledge base (KB) in a sequential recommender system [36]. In particular, the user can specify attribute-level or sequential preference at each session that are encoded using low-dimensional vectors.

The SceneRec tool [37] leverages GNNs to support scene-based recommender systems. Scenes represent sets of items that occur together in real-world situations. The underlying neural network is used to encode and propagate scene information by exploiting two different graphs, i.e., user-item and scene-based graph that are used to suggest relevant items by means of pairwise learning.

Lyu *et al.* [38] proposed RGRec, a rule-guided GNN to extract relation paths from the encoded entities, i.e., users, items, and their interactions. Given the initial graphs, a rule filtering mechanism is applied to identify long-range semantics among the aforementioned entities. RGRec computes the internal weights by exploiting rule learning techniques.

To the best of our knowledge, MORGAN is the first multi-purpose recommender system being built on top of a GNN to support the specification of both metamodels and models.

VII. Conclusion and future work

Modelers are facing an overload of information in their daily tasks, and this triggers the need for decent machinery to assist them in choosing suitable sources of information. Though various modeling frameworks are in place, there is still a lack of automated assistance which can help modelers ease the burden of the modeling activities. To this end, our proposed tool named MORGAN is a model recommender system that has been built on top of a graph neural network, with the ultimate aim of supporting modelers. We evaluated the performance of our tool on two real-world datasets using various quality metrics. The results demonstrate that MORGAN can provide relevant recommendations, including metaclasses and structured features for metamodels, as well as classes and class members for models.

For future work, we plan to further improve MORGAN by employing different encoding mechanisms on the input data. Furthermore, we suppose that link prediction techniques can be applied as an alternative strategy to complete models represented in a graph-based format. We plan to investigate the adoption of MORGAN for managing different kinds of models further than structural specifications as considered in this paper. Last but not least, we are going to curate a more well-defined dataset which, in the end, is expected to help MORGAN enhance its prediction performance.

Acknowledgments

The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223.

REFERENCES

[1] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, A. Pierantonio, and L. Iovino, "Automated classification of metamodel repositories: A machine learning approach," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Sep. 2019, pp. 272–282.

[2] P. T. Nguyen, D. Di Ruscio, A. Pierantonio, J. Di Rocco, and L. Iovino, "Convolutional neural networks for enhanced classification mechanisms of metamodels," *Journal of Systems and Software*, p. 110860, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121220302508

[3] G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean, J.-M. Jézéquel, T. Kühn, S. Mosser, H. Sahraoui, E. Syriani, D. Varró, and M. Weyssow, "Opportunities in intelligent modeling assistance," *Software and Systems Modeling*, vol. 19, no. 5, pp. 1045–1053, Sep. 2020. [Online]. Available: http://link.springer.com/10.1007/s10270-020-00814-5

[4] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An nlp-based architecture for the autocompletion of partial domain models," in *Advanced Information Systems Engineering*, M. La Rosa, S. Sadiq, and E. Teniente, Eds. Cham: Springer International Publishing, 2021, pp. 91–106.

[5] M. Weyssow, H. A. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," *CoRR*, vol. abs/2104.01642, 2021. [Online]. Available: https://arxiv.org/abs/2104.01642

[6] H. Bruneliere, J. Cabot, G. Dupé, and F. Madiot, "MoDisco: a Model Driven Reverse Engineering Framework," *Information and Software Technology*, vol. 56, no. 8, pp. 1012–1032, Aug. 2014. [Online]. Available: https://hal.inria.fr/hal-00972632

[7] MODELS2021, "Replication package for MORGAN," May 2021, original-date: 2021-05-13T10:42:53Z. [Online]. Available: https://github.com/MDEGroup/MORGAN

[8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[9] V. N. Ioannidis, A. G. Marques, and G. B. Giannakis, "Graph neural networks for predicting protein functions," in *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2019, pp. 221–225.

[10] C. W. Coley, W. Jin, L. Rogers, T. F. Jamison, T. S. Jaakkola, W. H. Green, R. Barzilay, and K. F. Jensen, "A graph-convolutional neural network model for the prediction of chemical reactivity," *Chem. Sci.*, vol. 10, pp. 370–377, 2019. [Online]. Available: http://dx.doi.org/10.1039/C8SC04228D

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[12] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 273–283. [Online]. Available: https://www.aclweb.org/anthology/P18-1026

[13] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 1802–1808. [Online]. Available: https://doi.org/10.24963/ijcai.2017/250

[14] S. Wu, W. Zhang, F. Sun, and B. Cui, "Graph neural networks in recommender systems: A survey," *CoRR*, vol. abs/2011.02260, 2020. [Online]. Available: https://arxiv.org/abs/2011.02260

[15] I. Maksimov, R. Rivera-Castro, and E. Burnaev, "Addressing cold start in recommender systems with hierarchical graph neural networks," 2020.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[18] K. Riesen and H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding*. USA: World Scientific Publishing Co., Inc., 2010.

[19] S. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. 40, pp. 1201–1242, 2010. [Online]. Available: http://jmlr.org/papers/v11/vishwanathan10a.html

[20] R. Clarisó and J. Cabot, "Applying graph kernels to model-driven engineering problems," in *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, ser. MASES 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–5. [Online]. Available: https://doi-org.univaq.clas.cineca.it/10.1145/3243127.3243128

[21] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, "GraKeL: A Graph Kernel Library in Python," *arXiv:1806.02193 [cs, stat]*, Mar. 2020, arXiv: 1806.02193. [Online]. Available: http://arxiv.org/abs/1806.02193

[22] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2539–2561, Nov. 2011.

[23] Önder Babur, "A labeled Ecore metamodel dataset for domain clustering," Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2585456

[24] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta, "Focus: A recommender system for mining api function calls and usage patterns," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1050–1060.

[25] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *CoRR*, vol. abs/1811.12808, 2018. [Online]. Available: http://arxiv.org/abs/1811.12808

[26] J. L. Hintze and R. D. Nelson, "Violin plots: A box plot-density trace synergism," *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/00031305.1998.10480559

[27] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

[28] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[29] A. Mora Segura and J. de Lara, "Extremo: An Eclipse plugin for modelling and meta-modelling assistance," *Science of Computer Programming*, vol. 180, pp. 71–80, Jul. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642319300644

[30] G. Dupont, S. Mustafiz, F. Khendek, and M. Toeroe, "Building Domain-Specific Modelling Environments with Papyrus: An Experience Report," in *2018 IEEE/ACM 10th International Workshop on Modelling in Software Engineering (MiSE)*, May 2018, pp. 49–56, iSSN: 2575-4475.

[31] E. Batot and H. Sahraoui, "A generic framework for model-set selection for the unification of testing and learning MDE tasks," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. Saint-malo France: ACM, Oct. 2016, pp. 374–384. [Online]. Available: https://dl.acm.org/doi/10.1145/2976767.2976785

[32] J. J. López-Fernández, J. S. Cuadrado, E. Guerra, and J. de Lara, "Example-driven meta-model development," *Software & Systems Modeling*, vol. 14, no. 4, pp. 1323–1347, Oct. 2015. [Online]. Available: http://link.springer.com/10.1007/s10270-013-0392-y

[33] T. Kuschke, P. Mäder, and P. Rempel, "Recommending Auto-completions for Software Modeling Activities," in *Model-Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, Eds. Berlin, Heidelberg: Springer, 2013, pp. 170–186, 00015.

[34] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Domobot: A bot for automated and interactive domain modelling," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi-org.univaq.clas.cineca.it/10.1145/3417990.3421385

[35] X. Xian, L. Fang, and S. Sun, "Regnn: A repeat aware graph neural network for session-based recommendations," *IEEE Access*, vol. 8, pp. 98 518–98 525, 2020.

[36] B. Wang and W. Cai, "Knowledge-enhanced graph neural networks for sequential recommendation," *Information*, vol. 11, no. 8, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/8/388

[37] G. Wang, Z. Guo, X. Li, D. Yin, and S. Ma, "Scenerec: Scene-based graph neural networks for recommender systems," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra, Eds. OpenProceedings.org, 2021, pp. 397–402. [Online]. Available: https://doi.org/10.5441/002/edbt.2021.41

[38] X. Lyu, G. Li, J. Huang, and W. Hu, "Rule-guided graph neural networks for recommender systems," in *The Semantic Web – ISWC 2020*, J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, and L. Kagal, Eds. Cham: Springer International Publishing, 2020, pp. 384–401.