Software and Systems Modeling
https://doi.org/10.1007/s10270-023-01102-8

SPECIAL SECTION PAPER

MORGAN: a modeling recommender system based on graph kernel

Claudio Di Sipio[1] · Juri Di Rocco[1] · Davide Di Ruscio[1] · Phuong T. Nguyen[1]

Received: 1 April 2022 / Revised: 11 February 2023 / Accepted: 6 March 2023

Abstract
Model-driven engineering (MDE) is an effective means of synchronizing among stakeholders, thereby being a crucial part of the software development life cycle. In recent years, MDE has been on the rise, triggering the need for automatic modeling assistants to support metamodelers during their daily activities. Among others, it is crucial to enable model designers to choose suitable components while working on new (meta)models. In our previous work, we proposed MORGAN, a graph kernel-based recommender system to assist developers in completing models and metamodels. To provide input for the recommendation engine, we convert training data into a graph-based format, making use of various natural language processing (NLP) techniques. The extracted graphs are then fed as input for a recommendation engine based on graph kernel similarity, which performs predictions to provide modelers with relevant recommendations to complete the partially specified (meta)models. In this paper, we extend the proposed tool in different dimensions, resulting in a more advanced recommender system. Firstly, we equip it with the ability to support recommendations for JSON schema that provides a model representation of data handling operations. Secondly, we introduce additional preprocessing steps and a kernel similarity function based on item frequency, aiming to enhance the capabilities, providing more precise recommendations. Thirdly, we study the proposed enhancements, conducting a well-structured evaluation by considering three real-world datasets. Although the increasing size of the training data negatively affects the computation time, the experimental results demonstrate that the newly introduced mechanisms allow MORGAN to improve its recommendations compared to its preceding version.

Keywords Model-driven engineering · Recommender systems · Graph kernels

1 Introduction

The deployment of model-driven engineering (MDE) techniques necessitates advanced tools to facilitate various modeling activities [1,2]. Among others, there is the need to specify metamodels, models and the development of model analysis and management operations. Nevertheless, existing tools such as those that are based on Eclipse EMF[1] normally offer only canonical functionalities, i.e., drag-and-drop, specification of graphical components, auto-completion, and they do not support context-related recommendations, which may come in handy for modelers to complete their tasks. In this respect, intelligent modeling assistants (IMAs) [3] have been recently proposed to support modelers during their daily activities. Most of the existing tools employ technologies such as neural networks and NLP techniques to automatize the whole design process [4,5]. Altogether, this aims to facilitate the completion of metamodels by providing modelers with insightful artifacts, such as attributes or relationships. Nonetheless, there are still open challenges to be tackled, e.g., offering a convenient way to specify metamodels and models, covering different application domains.

In our previous work, we developed MORGAN [6]—a MOdeling Recommender system based on a Graph neurAl Network to support the completion of both models and metamodels. MORGAN has been evaluated using two real-world

Communicated by Shiva Nejati and Daniel Varro.

✉ Davide Di Ruscio
davide.dirsucio@univaq.it

Claudio Di Sipio
claudio.disipio@univaq.it

Juri Di Rocco
juri.dirocco@univaq.it

Phuong T. Nguyen
phuong.nguyen@univaq.it

1 Universitádegli studi dell'Aquila, L'Aquila, Italy

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

datasets composed of models and metamodels. The experiment results showed that our proposed tool can provide relevant recommendations related to model classes and class members, as well as relevant metaclasses and structural features.

In this work, we improve the existing system to yield an advanced version, by supporting the completion of models expressed in Javascript Object Notation (JSON).[2] Such a format is used to represent data in a structured way, fostering the interchangeability of valuable information. Even with the introduction of JSON schema meta-language [7], it is still essential to assist developers during their specification. In this respect, a recent work presents a bridge between the JSON schema and MDE metamodels, suggesting that MDE techniques can come in handy in designing such models [8]. Thus, we introduce the support for JSON schema completion by relying on a tailored parser to encode these specifications in a graph-based structure. Furthermore, we overhaul the underlying recommendation engine by introducing (i) a lemmatization preprocessing step to improve MORGAN 's encoding; and (ii) an enhanced graph kernel function by relying on a modified version of the Vertex Histogram technique [9] that exploits a frequency matrix. To enable the recommendation on the new dataset, we mined JSON schema from GitHub repositories by using a well-defined crawler. Afterward, the relevant features are represented in a graph-based structure to encode information represented by the model, e.g., relationships among defined entities, the name of the attributes, to list a few. Such an encoding process exploits a set of tailored parsers that produce textual files from different formats used to represent the modeling artifacts of interest. MORGAN eventually suggests relevant artifacts including the JSON schema elements, demonstrating the generalizability of the system. Furthermore, we tested the performance of MORGAN on ModelSet, a recent and publicly available dataset composed of a large number of modeling artifacts [10]. Using the provided interfaces, we filter the original data to obtain two different datasets composed of Ecore metamodels and UML class diagrams. The aim is to investigate the system's scalability by increasing the training data size.

To study the proposed mechanisms, we conduct an empirical evaluation to compare the new results with those presented in our previous work, exploiting several quality metrics. Our findings demonstrate that the introduced boosting schemes contribute to improving the overall accuracy, though MORGAN suffers from scalability issues when a larger number of training artifacts are considered. In this respect, the newly conceived system is capable of providing modelers with a practical means to perform their tasks even considering a different application domain.

To this end, compared to our previous work [6], this extension makes the following contributions:

- We improve the underlying recommendation engine by adopting a well-defined graph kernel;
- Three additional datasets have been considered to test the generalizability and scalability of MORGAN ;
- A protocol to download model representation from GitHub repositories is defined;
- A replication package is made available to facilitate future research.[3]

Structure. Section 2 presents an illustrative example to motivate our proposed approach as well as an overview of the graph kernel algorithm. The main components of MORGAN are described in Sect. 3. In Sect. 4, we explain in detail the methods and materials to conduct an empirical evaluation. Section 5 reports and analyzes the obtained results. Afterward, possible threats to the validity of our findings are listed in Sect. 6. We review relevant work and conclude the paper in Sects. 7 and 8, respectively.

2 Motivations and background

Section 2.1 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 2.2 presents an overview of graph neural networks (GNNs) and graph kernel similarity and how they can be used to cope with the identified issues.

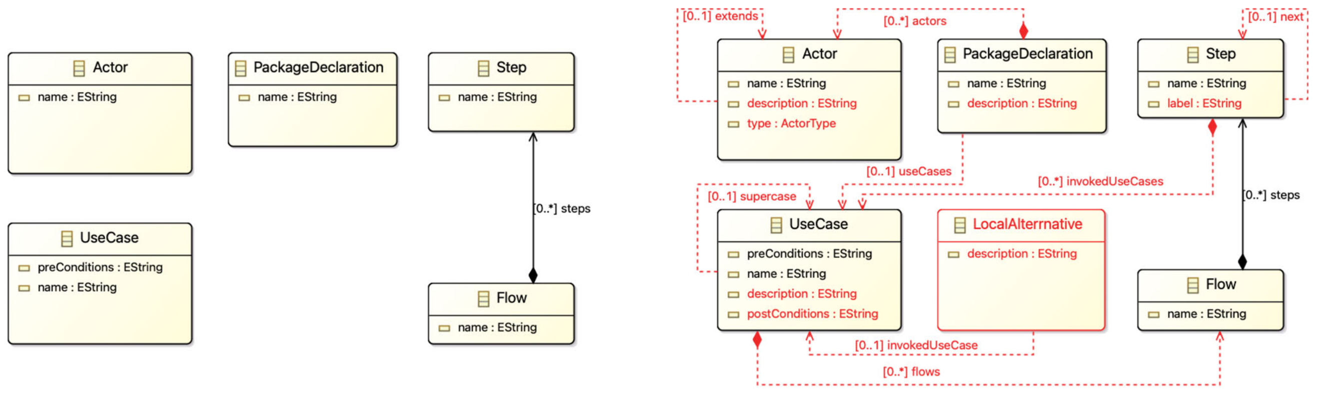
2.1 Motivating example

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, he/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. Fig. 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. 1a defines the key concepts to represent a UML *UseCase*, i.e., *Actor*, *Step*, and *Flow*. Figure 1b represents the final version of the corresponding metamodel, where the *PackageDeclaration* entity is completed with *use*

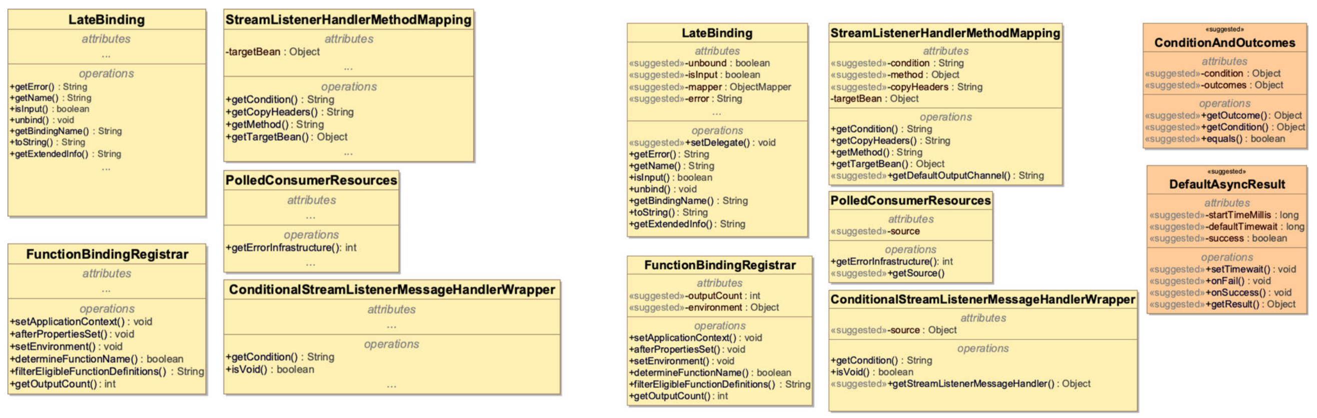
[2] https://www.w3.org/TR/json-ld11/.

[3] https://github.com/MDEGroup/MORGAN.



(a) Partial metamodel (b) Complete metamodel

Fig. 1 The illustrative UmlDSL


LateBinding
StreamListenerHandlerMethodMapping
PolledConsumerResources
FunctionBindingRegistrar
ConditionalStreamListenerMessageHandlerWrapper
LateBinding
StreamListenerHandlerMethodMapping
ConditionAndOutcomes
PolledConsumerResources
DefaultAsyncResult
FunctionBindingRegistrar
ConditionalStreamListenerMessageHandlerWrapper

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

Fig. 2 The explanatory SpringBoot model

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. 1a, 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. 1b, 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.

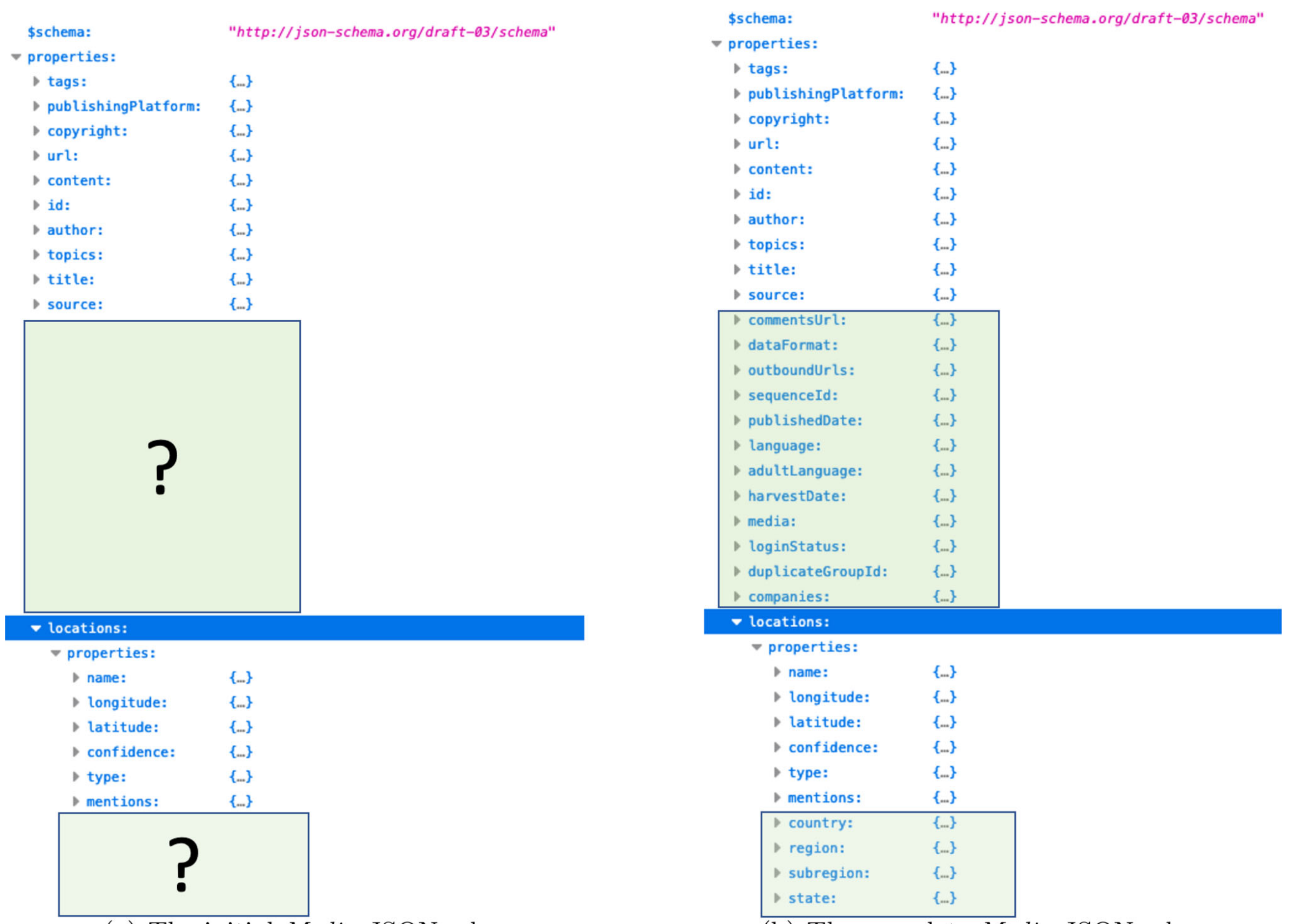
Modeling assistant. We make use of the MoDisco framework[4] to extract models from compilable Eclipse projects. As an example, we show in Fig. 2a an excerpt of the Java model of the Spring bootproject[5] parsed by MoDisco . The model conforms to the MoDisco Java metamodel, covering the full Java language 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. 2a 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

[4] https://www.eclipse.org/MoDisco/.

[5] https://spring.io/projects/spring-boot.



```
$schema:              "http://json-schema.org/draft-03/schema"
▾ properties:
  ▸ tags:               {…}
  ▸ publishingPlatform: {…}
  ▸ copyright:          {…}
  ▸ url:                {…}
  ▸ content:            {…}
  ▸ id:                 {…}
  ▸ author:             {…}
  ▸ topics:             {…}
  ▸ title:              {…}
  ▸ source:             {…}
          ?
  ▾ locations:
    ▾ properties:
      ▸ name:           {…}
      ▸ longitude:      {…}
      ▸ latitude:       {…}
      ▸ confidence:     {…}
      ▸ type:           {…}
      ▸ mentions:       {…}
            ?
```

(a) The initial *Media* JSON schema

```
$schema:              "http://json-schema.org/draft-03/schema"
▾ properties:
  ▸ tags:               {…}
  ▸ publishingPlatform: {…}
  ▸ copyright:          {…}
  ▸ url:                {…}
  ▸ content:            {…}
  ▸ id:                 {…}
  ▸ author:             {…}
  ▸ topics:             {…}
  ▸ title:              {…}
  ▸ source:             {…}
  ▸ commentsUrl:        {…}
  ▸ dataFormat:         {…}
  ▸ outboundUrls:       {…}
  ▸ sequenceId:         {…}
  ▸ publishedDate:      {…}
  ▸ language:           {…}
  ▸ adultLanguage:      {…}
  ▸ harvestDate:        {…}
  ▸ media:              {…}
  ▸ loginStatus:        {…}
  ▸ duplicateGroupId:   {…}
  ▸ companies:          {…}
  ▾ locations:
    ▾ properties:
      ▸ name:           {…}
      ▸ longitude:      {…}
      ▸ latitude:       {…}
      ▸ confidence:     {…}
      ▸ type:           {…}
      ▸ mentions:       {…}
      ▸ country:        {…}
      ▸ region:         {…}
      ▸ subregion:      {…}
      ▸ state:          {…}
```

(b) The complete *Media* JSON schema

Fig. 3 The explanatory Apache Streams Provider JSON Schema

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. 2b, 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.

JSON schema assistant. Similarly, modeling assistants can be employed to complete a partial JSON schema with relevant properties. Although they are artifacts of different nature, Colantoni et al. [8] investigate the potential benefits of mapping JSONware technical space to Modelware one, with the aim of making use of languages and documents exchangeable. Their approach supports language engineers, domain experts, and tool providers in editing, validating, and generating tool support with enhanced capabilities for JSON schemas and their documents. The results show that it is possible to derive editing and validation support based on model-driven technologies for JSON Schema-based languages. Therefore, modeling assistants could be employed to support the completion of such kinds of artifacts. Figure 3 shows an explanatory example with the *Apache Streams provider*[6] JSON schema to describe a *Media* entity. Given the partial schema depicted in Fig. 3a, we mimic the situation where the modeler has specified a set of initial 11 properties, e.g., tags, copyright, url, author to name a few. At this point, an automated system might be used to fill the partial JSON schema with additional items, such as new properties at schema level or nested ones. In Fig. 3b, we devise a possible set of recommendations that includes additional properties, e.g., *dataFormat*, *language*, *adultLanguage*. Furthermore, the *locations* structural property can be enriched

[6] https://streams.apache.org/.

with concepts that can increase the expressiveness of the whole schema, i.e., country, region, subregion, and state.

The presented use cases raise the need for a model assistant to support the completion of partially defined metamodels, models, and JSON schema. 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.

2.2 Graph kernel similarity

A graph is often represented as an adjacency matrix $\mathbf{A}$ of size $N \times N$, where N is the number of nodes. If each node is characterized by a set of M features, then a dimension of feature matrix X is $N \times M$. Graph-structured data are complex, and thus, it brings a lot of challenges for existing machine learning algorithms.

Given a graph, the prediction phase can be realized by following two different strategies, i.e., *link-level prediction* or *node-level classification.* The former requires to represent the relationship among nodes in graph and predict if there is a connection between two entities. In latter, the task is to understand node embedding for every node in a graph by looking at the labels of their neighbors. With *graph-level classification*, the entire graph needs to be classified into suitable categories.

Different kinds of algorithms can be used to produce recommendations, i.e., vector-space embedding [11] and graph kernel [12]. 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 [13].

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:

$$k(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle_{\mathcal{H}} \tag{1}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product defined into a Hilbert Space H.

A graph kernel computes the similarity between two graphs by adopting several different strategies. Among these, the Weisfeiler–Lehman optimal assignment algorithm (WLOA kernel hereafter) [14] is built on top of existing graph kernels and it is inspired by the Weisfeiler–Lehman test of graph isomorphism [15].

In the previous version of MORGAN [6], we opted for the WLOA technique to measure the similarity as it offers a linear complexity. The algorithm replaces each vertex with a multiset representation consisting of the original one plus neighbors' features. Given two graphs G and G', the WLOA kernel is defined as follows:

$$k(G, G') = K^{k}_{\mathfrak{B}}(V, V') \tag{2}$$

where k is the employed base kernel, defined below:

$$k(v, v') = \sum_{i=0}^{h} \delta(\tau_i(v), \tau_i(v')) \tag{3}$$

where $\tau_i(v)$ is the label of node v at the end of ith iteration of the Weisfeiler–Lehman algorithm. However, for this kernel recommendation time is strongly bounded by the nature of the input data.

In this extension, we consider a different class of kernel function, namely the Vertex Histogram [9], to analyze to what extent the adoption of a different technique can reduce the whole prediction time.

In practice, given a pair of graphs G and G', let f, f' be the vertex histograms of the two graphs. The Vertex Histogram kernel is computed according to the following formula:

$$k(G, G') = \langle \mathbf{f}, \mathbf{f}' \rangle \tag{4}$$

where $\mathbf{f}$, $\mathbf{f}'$ are the vertex label histograms obtained by applying the mapping function $\ell : \mathcal{V} \rightarrow \mathcal{L}$ that assigns labels to the vertices of the graphs.[7]

Even though the underlying structure is not considered by the employed kernel, our empirical results demonstrate that MORGAN achieves better performances in terms of accuracy and time computation compared to the Weisfeiler–Lehman for this specific task. Therefore, we employ 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 Proposed approach

This section describes in detail the proposed approach, whose architecture is depicted in Fig. 4. Starting from a partial model, MORGAN uses tailored model parsers to

[7] This process is internally performed by the Python implementation offered by Grakel.

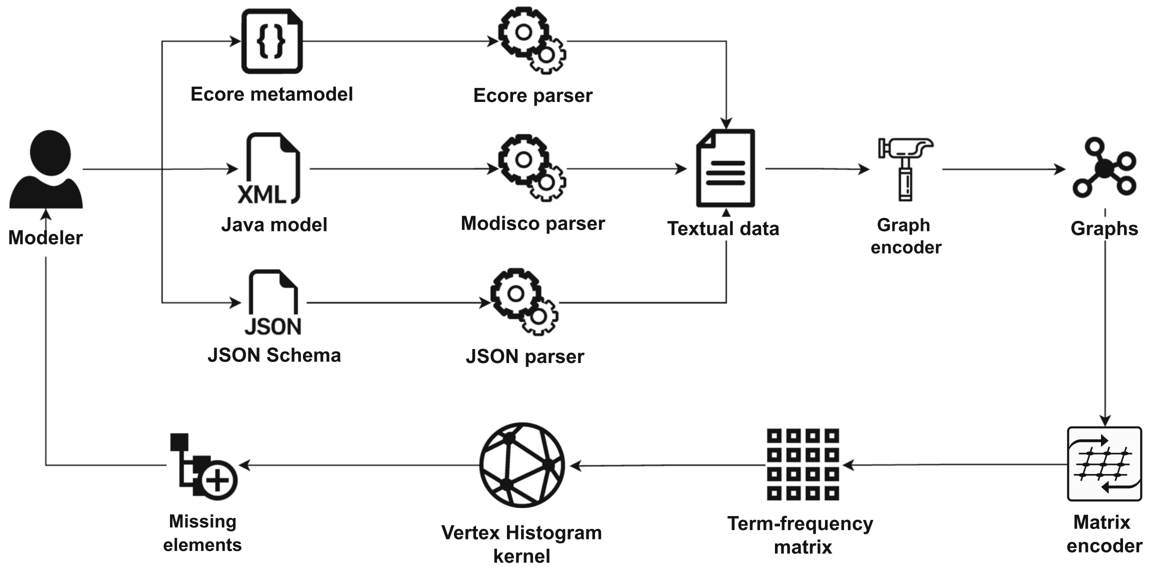


Fig. 4 The MORGAN architecture

extract relevant information in textual data format. Alongside the two independent parsers developed in the original work, i.e., one for metamodels and the other for models, we conceive two additional parsers tailored for JSON schema and UML models. The NLP module executes the encoding to generate graphs that can be used by the underlying graph kernel engine. In this work, we enrich this component with a lemmatization strategy to increase the amount of valuable information. Furthermore, we adopted the Vertex Histogram kernel as the underpinning recommendation engine which is based on item frequency. As the final recommendations, MORGAN returns the missing elements of the artifact under development, i.e., either a metamodel or a model.[8] We describe the constituent components as follows.

3.1 Parsers

This component extracts relevant information from the input artifact by using four different parsers, i.e., *Ecore parser* for metamodels, *MoDisco parser* for MoDisco models, *UML parser* for UML models, and a standard *JSON parser*[9] for JSON schema. Because metamodels and models are expressed in the XMI or Ecore format, the first three parsers rely on EMF utilities[10] to extract their information. In contrast, the JSON schema parsers rely on utilities that embody the standard reference schema.

It is worth noting that this component is compliant with a well-structured encoding scheme. In such a way, the proposed tailored parsers follow the same structure during the data gathering phase, i.e., they are agnostic from the underlying model artifact. The generic encoding scheme is defined as follows:

$$\text{instance id}, (\langle \text{relationship tuple} \rangle)^* \tag{5}$$

where the instances are represented by their identifier (e.g., name) and the relationships are tuples that encode related concepts depending on the notation. It is worth noting that a tuple can be composed of two or three elements, according to the considered modeling artifact.

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 metaclass name identifies the corresponding metaclass instance while each relationship is represented as a *triple* defined as follows:

$$\text{metaclass-name } (\langle \text{Name, Type, [reference} \mid \text{attribute]} \rangle)^* \tag{6}$$

where Name is the name of the structural feature, Type characterizes the type element, e.g., EString or EInt, and Relation identifies the type of relation between the element and the class, i.e., attribute or reference. Following the aforementioned scheme, the Actor metaclass described in Fig. 1a is encoded as `Actor (name, EString, attribute)`. These elements improve MORGAN 's recommendation capabilities as they enable the modeler to add further information to the partial model.

[8] To facilitate the presentation, the common name *artifact* refers to all kinds of considered items, namely *metamodel*, *model*, and *JSON schema*, unless otherwise explicitly stated.

[9] https://github.com/google/gson.

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

MoDisco model parsing. MoDisco parser is used to extract valuable data from models. In particular, similar to *Ecore parser*, *MoDisco parser* explores *xmi* trees to elicit valuable elements, i.e., each MoDisco model is represented as a list of *Java classes* followed by *method definition*s and *field declaration*s. The class name represents the instance identifier while each relationship is represented as a *triple* defined as follows:

$$\text{class-name } (\langle \text{Name, Return type, [method} \mid \text{field]}\rangle)^* \tag{7}$$

where Name is the method or field name, Return type is the method or field type, and Relation identifies the type of relation between the class members, i.e., method or field, and the class. For instance, the signature of the getError method belonging to *LateBinding* class depicted in Fig. 2a is translated as `LateBinding (getError, EString, method)`.

UML model parsing. To extract the same type of information from models belonging to ModelSet, a UML parser has been conceived by exploiting EMF utilities. It is worth noting that the structure of the parser is similar to the previous ones, i.e., it navigates the tree structure of the UML models given as input. To be consistent with the parser requirements, we elicit *properties* and *operations* for each UML class. The encoding scheme for a single UML class is the following:

$$\text{class-name } (\langle \text{Name, Type}\rangle)^* \tag{8}$$

where Name is the name of the property or operation of the class while Type can be operation or property. For instance, a UML class named Teacher with the property name and the operation setMark is translated into `Teacher (name, property) (setMark,operation)`.

JSON schema parsing. Similarly to the metamodels that provide the notation to express models, JSON schema [7] is a dedicated language to define and validate JSON documents. A JSON schema document consists of two main types of schemas, i.e., *BooleanSchema* and *ObjectSchema*. In the scope of this paper, we focus on the latter that defines the hierarchical structure of the schema, including (i) the *Type* that could be a *SimpleType* or *object*, and (ii) a list of *Keyword* elements that defines the properties of the object. Each element of this list contains a *KeySchemaPair* that explicitly defines the structural features of an object or the primitive type. In this case, we opt for ObjectSchema name as the instance identifier, whereas the list of properties identified by *KeywordsPair* is used to identify the relationship tuples. We encoded such data as following:

$$\text{ObjectSchema-name } (\langle \text{name, [object} \mid \text{int} \mid \text{string} \mid \ldots]\rangle)^* \tag{9}$$

If Type is a primitive type, e.g., string or Boolean, Keywords are mapped to attributes, otherwise they are contained objects. In such a way, the JSON schema parser is used to extract pairs consisting of ObjectSchema and contained properties. This parser is capable of extracting the same type of information compared to the aforementioned ones. An excerpt of the encoding of *locations* highlighted in Fig. 3a can be represented as `Locations (name, string)(longitude, int)(mentions, object)`.

3.2 Graph encoder

The next step is to build graphs from textual files produced by the parsing phase. In the former version of the tool [6], the artifact *Encoder* component applied a standard NLP pipeline composed of three main steps, i.e., stop-words removal, string cleaning, and stemming. In this extension, we employ the lemmatization strategy instead of stemming by adopting the well-founded algorithm based on WordNet[11] The rationale behind this choice is that the former strategy is a process for removing the commoner morphological and inflexional endings from English words [16]. However, this could lead to incorrect meaning and spelling since the semantic is not considered at all. Meanwhile, lemmatization considers the context and extracts the lemma of each word, i.e., its base form. To enable this process, a large thesaurus of English terms is needed since the lemmatizer component involves the morphological analysis of each word. In the scope of this paper, we make use of the WordNet Lemmatizer utility provided by the *nltk* Python library[12] to extract the *lemma* from each analyzed term. Afterward, a corpus of words is created from scratch by inserting the preprocessed 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,[13] a Python library that creates nodes and edges considering the structure of the parsed model. According to the format shown in Eq. 5, each model is represented by a list of connected graphs in which each class is linked with corresponding elements.

An explanatory graph obtained by the Encoder component is depicted in Fig. 5. It is an encoding fragment of the metaclass Actor shown in Fig. 1b. In particular, the structure of the model is encoded by adding edges among the class Actor and the stemmed version of its attributes and references namely Name, Description, Type, and Extends, though such type of graph does not resemble the semantic relationships occurring

[11] https://wordnet.princeton.edu/.

[12] https://www.nltk.org/_modules/nltk/stem/wordnet.html.

[13] https://networkx.org/.

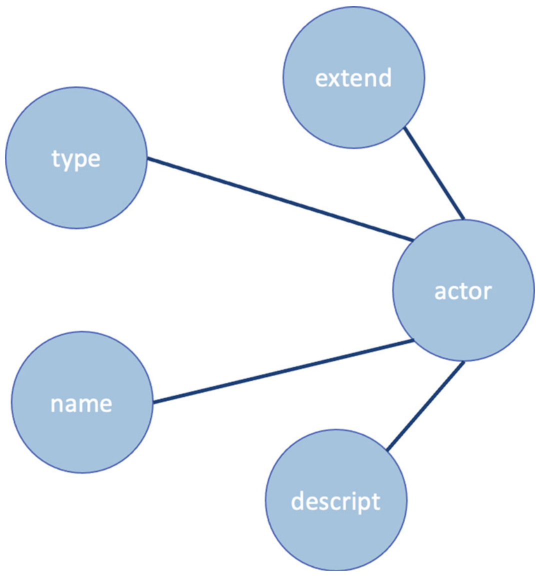


Fig. 5 Example of a stemmed graph

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 underpinning model. It is worth mentioning that the same format is kept to perform the final recommendations, thus delivering additional information about the type of the model element to the user.

3.3 Matrix encoder

At this point, the Matrix Encoder component counts the occurrences of the items belonging to the obtained graphs. Each item has three required features: i.e., adjacency_matrix, node_labels, and edge_labels. The first feature is required to ensure that the graph is valid from the structural point of view. The second and third features are not mandatory in some certain contexts. For instance, e.g., edge_labels is not required in case the kernel algorithm does not employ them in the recommendation phase.[14] To feed the graph kernel, Matrix Encoder assigns a unique ID to each node of the graph to build a term-frequency matrix. In such a way, the most frequent items are considered at recommendation time. This component eventually calculates the sum of products between frequencies of common occurrences by employing the diagonal method to perform the dot product operation. It is worth mentioning that this strategy speeds up the diagonal calculation to reduce the computation time.

3.4 Vertex histogram kernel

At this point, the underpinning 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 [17]. We compute graph similarity by comparing the input model with all the elements in the training set. As discussed in Sect. 2, we replaced the WLOA kernel [14] employed in our previous work with the Vertex Histogram kernel strategy as described in Sect. 2.2. It is important to remark that we added the Matrix encoder component that produces an adjacency matrix given the produced graphs. In such a way, the structure of the graphs is preserved, thus enabling the computation of the kernel similarity.

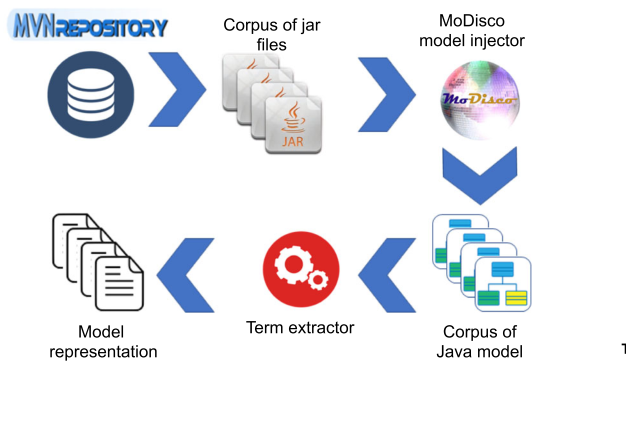
The rationale behind the selection of Vertex Histogram is as follows. The implemented version of the algorithm is linear in the number of vertices of the graph.[15] Moreover, through an empirical evaluation, we realized that it is much more timing efficient than the graph kernel adopted in the former version of MORGAN, i.e., WLOA strategy. Thus, Vertex Histogram has been chosen to compute the similarities among the nodes in graphs.

The employed kernel eventually retrieves an ordered list of modeling artifacts stored in the training set ranked by similarity scores. These values are computed by exploiting the Vertex histogram algorithm as described in Eq. 4. In practice, given the context of the modeler encoded as a graph, MORGAN compares it with the graphs extracted from the training set. At the end of the process, a similarity score is assigned for each comparison and a ranked list of graphs is produced. The top-5 similar elements are extracted from this 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. It is worth noting that the whole process is adopted for each type of model artifact, i.e., metamodels, Java models, and JSON schema. This means that there are three different training processes in which MORGAN learns a specific set of features needed to recommend the proper items.

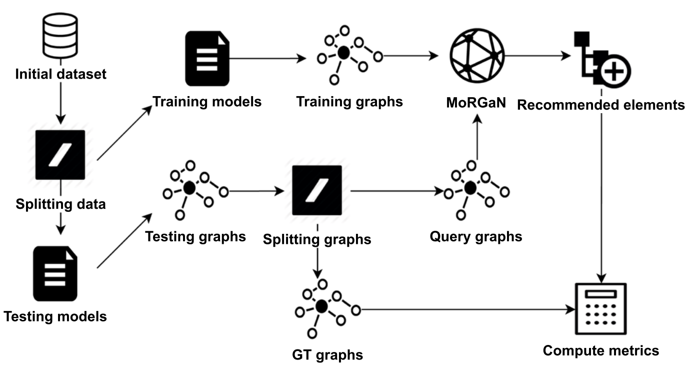
Being built on these components, MORGAN provides recommendations including both metamodels, models, and JSON schema. The succeeding subsection introduces an explanatory example to show to which extent MORGAN is useful for a metamodeling task.

[14] In the scope of the paper, we do not employ the edge_labels feature since homogeneous graphs are considered.

[15] https://ysig.github.io/GraKeL/0.1a8/kernels/vertex_histogram.html?highlight=vertex%20histogram.



(a) The process to generate the D_α dataset



(b) The evaluation process

Fig. 6 D_α dataset creation and overall evaluation process

Table 1 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

3.5 Explanatory example

Table 1 shows the suggested structural features for the UMLDSL metamodel depicted in Fig. 1a. 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 1. 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.

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.

4 Evaluation

We describe in detail the research objectives and the experimental configurations used to study MORGAN 's performance. First, the research questions that we address in this paper are presented in Sect. 4.1. Afterward, in Sect. 4.2 we describe the datasets used in the evaluation. Finally, the validation methodology and evaluation metrics are detailed in Sects. 4.3 and 4.4, respectively.

4.1 Research questions

The following research questions are addressed to study the new version of MORGAN , comparing it with the previous one [6].

- **RQ$_1$**: *Does the preprocessing step contribute to a performance gain of MORGAN ?* We investigate whether the introduced preprocessing augmentation improves the overall performances in terms of identified metrics, i.e., success rate, precision, recall, and F-measure.
- **RQ$_2$**: How does the vertex histogram kernel function impact on the computational efficiency? In an attempt to extend the MORGAN tool, we employed a tailored graph kernel based on the term-frequency matrix as the underlying recommendation engine. This research question aims to validate if the proposed mechanism helps MORGAN reduce the time required to perform the recommendations.
- **RQ$_3$**: *How effective is MORGAN at recommending JSON schema elements?* To assess if the tool is able to support different application domains, we evaluate by considering a curated dataset composed of JSON schema models.

Such type of data is widely used in MDE, and thus we suppose that the capability to work with JSON will allow MORGAN to gain popularity in real-world scenarios.

– **RQ$_4$**: *How is MORGAN 's performance changed when working on ModelSet, a benchmark dataset of metamodels and UML models?* To further study MORGAN 's capabilities in recommending modeling artifacts, we considered two additional datasets extracted from ModelSet[10], a recently collected set of metamodels and models. In such a way, we can measure how the size of the data affects the tool's overall performance.

4.2 Datasets extraction

Recent studies in the domain provide large, curated collections of models and metamodels, leveraging to support ML-based tasks, e.g., classification of models and prediction of relevant modeling artifacts. In this extension, we consider five different datasets, which are named D_α, D_β, D_γ, D_δ, and D_ϵ. The first two datasets, i.e., D_α, and D_β were already used to evaluate the former version of MORGAN [6], and we re-used them to evaluate the fine-tuned version of MORGAN proposed in this extension. Furthermore, with the aim of showcasing the extensibility of our approach, we make use of three additional datasets, namely D_γ, D_δ, and D_ϵ. The first one is composed of JSON schema crawled from GitHub. Meanwhile, D_δ and D_ϵ have been extracted from ModelSet [10], a curated collection of models and metamodels. The selected datasets are explained in detail as follows.

✧ Concerning D_α, we extracted model representations of popular Java projects stored in the *Maven repository*[16] to build the D_α dataset. The whole process to obtain the required data is depicted in Fig. 6a. 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.[17] 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. In such a way, we aim to build a curated collection of models that share common features as much as possible. Having such similarities could improve the overall accuracy even though this cannot be granted at the beginning of the process. Figure 7 shows the statistics for the extracted models in D_α. In particular, the x, y, and z axes correspond to the number of classes, the number of methods, and the number of fields of the mined artifacts, respectively. Moreover, the colors of dots are used to represent the categories. It is evident that most of the models

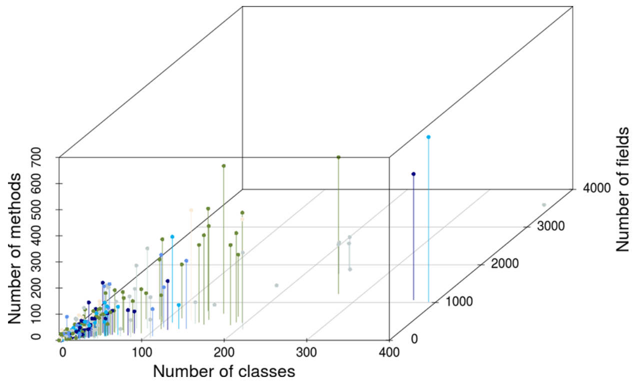


Fig. 7 D_α features

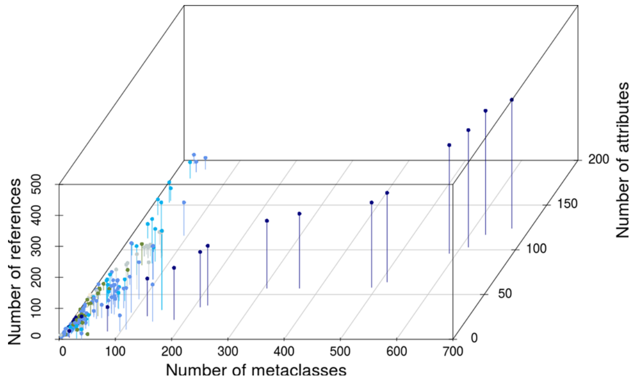


Fig. 8 D_β features

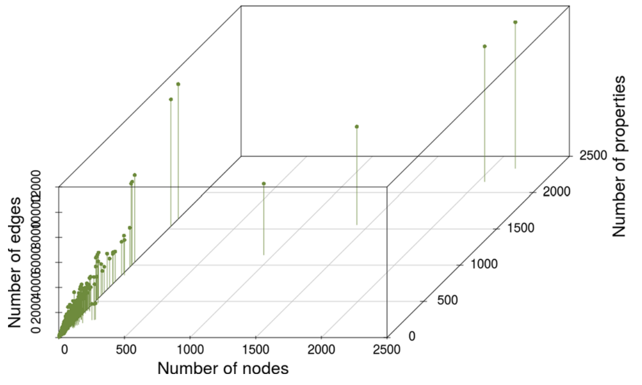


Fig. 9 D_γ features

contain a small number of methods and classes, i.e., lower than 300 and 200, respectively. There is only one model with more than 1,000 fields, 390 classes, and 690 methods. The corpus of JAR files has been collected by employing Beautiful Soup,[18] 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.,

[16] https://mvnrepository.com/.

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

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

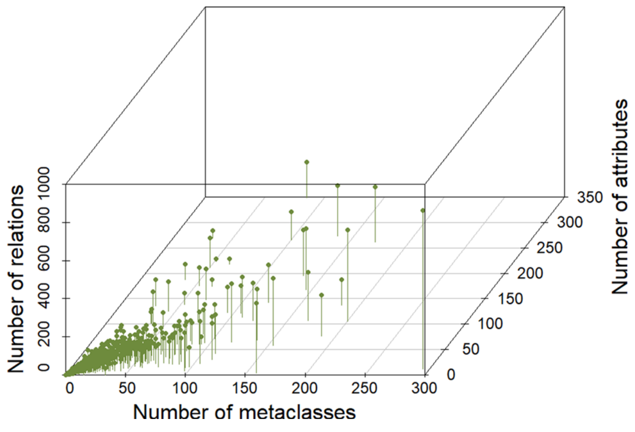


Fig. 10 D_δ features

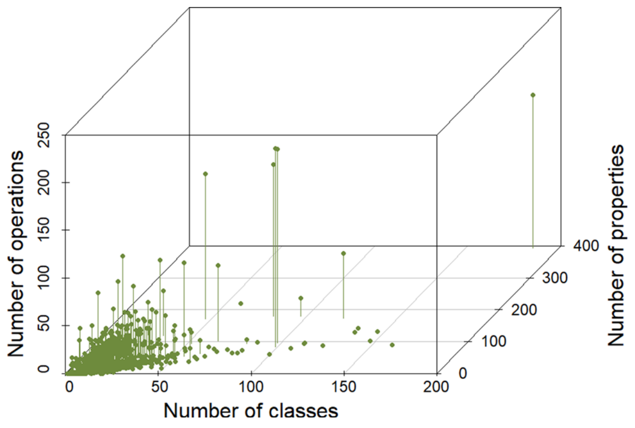


Fig. 11 D_ϵ features

classes, methods, and fields. Finally, a model parser is used to represent the model as defined in Sect. 3.1. In the end, we collected a set of 581 unique model representations from the MVN Repository belonging to the top categories.

✧ To generate D_β, starting from an original set consisting of 555 labeled metamodels with nine different categories [18], we extracted metaclasses, attributes, and references from each *Ecore* file using the Eclipse EMF utilities. Moreover, different quality filters have been also applied on the data, attempting to improve MORGAN 's performance. In particular, we removed metaclasses having 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 eliminated metamodels belonging to the Bug category, which has only eight metamodels. Possible duplicate classes are also excluded to avoid any bias. The final dataset consists of the following eight categories: *Build*, Conference, Office, PetriNets, Request, SQL, BibTex, and emphUML. Figure 8 reports a summary related to the characteristics of D_β including the number of metaclasses (the x axis), the number of references (the y axis), and the number of attributes (the z axis). Like in Fig. 7, the category of metamodels is represented using colors. Though we do not directly employ the category to produce recommendations, it includes similar metamodels which help to represent the application domain.

✧ As discussed in Sect. 2.1, the definition of a JSON schema falls under the umbrella of modeling activity. In the light of such rationale, we introduced an additional dataset, i.e., D_γ, consisting of JSON schema mined from GitHub. Due to limitations imposed by the GitHub REST API [19], we mined a publicly-available GitHub archive dataset freely accessible in Google BigQuery [20]. First, we query metadata searching for files with .json extension across the repositories in Ref. [20], restricting the search to .json files whose pathname contains the string schem. The rationale behind it is twofold: (i) it limits the amount of data processed by the query on the GitHub archive dataset to avoid exceeding the free monthly per user quota (1TB) [20]; (ii) we aim to infer insights on .json files by its pathname. We then filtered out dangling entries, i.e., pairs where pathnames point to .json files that no longer exist, and existing .json files that do not contain the $ *schema* keyword. Then, *duplicates removal*, *parsing*, and *conformance check* are performed.

The duplicates removal step computes a hash value for every mined file based on its content. The obtained hashes are used to build a hashmap where the key is the hash itself, and the value is the corresponding file. If a duplicated key occurs in the map, we assume that the corresponding .json file is a duplicate and we drop it. We distinguish between *parsing* and *conformance check* steps. The former checks if the JSON file is correctly serialized, while the latter validates the conformance of a schema considering its meta-schema definition.[19] Finally, we collected 2,872 distinct and valid JSON schemes, whose features are shown in Fig. 9.

✧ ModelSet is a collection of 5,466 Ecore metamodels and 5,120 UML models manually labeled by the authors [10]. This curated collection has been used in practice to enhance the performance of an existing search engine for models, namely MAR [21]. As shown in our previous work [22], the quality of the input data plays a key role in the performance of many model assistants. In particular, curated datasets with more similar metamodels allow recommender systems to improve their prediction performance, even if the size of such datasets is smaller than that of those randomly collected. Therefore, we make use of the provided Python API[20] to filter the initial list of artifacts by a tailored ModelSet query with the following parameters:

```
remove-duplicates = True,
remove-categories = ['dummy', 'unknown
    '],
min-occurrences-per-category = 7,
```

[19] https://json-schema.org/specification.html.

[20] https://github.com/modelset/modelset-py.

Table 2 Configuration settings

Configuration	Lemmatizer	Vertex histogram
C_1	✗	✗
C_2	✓	✗
C_3	✗	✓
C_4	✓	✓

```
languages = ['english']
```

Eventually, we extracted D_δ and D_ϵ and their features are shown in Figs. 10 and 11, respectively. In particular, we obtained 1247 metamodels and 849 UML models extracted from ModelSetusing the aforementioned quality filter. For all considered 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.

4.3 Settings

In the original study [6], we assessed the prediction performance of MORGAN by resembling the behaviors of a modeler[21] working at different stages of a modeling project m, by involving different configurations during the experiments [23]. To this end, we split m and use the rest as the modeler's context by varying two parameters, i.e., the number of considered classes and the number of the corresponding structural features. Starting from these definitions, we create four configurations to simulate different stages of modeling, i.e., from an initial specification to a mature one. We realized that MORGAN is capable of assisting the modeler in all considered scenarios, and the best accuracy scores were obtained when a mature context is considered [6]. In this work, we focus on assessing the contributions of the novelties introduced at the level of the recommendation engine, i.e., the additional preprocessing step and the vertex histogram kernel function. In particular, we are interested in measuring the contribution of each component separately and comparing the achieved results with the former version of MORGAN .[22] Table 2 summarizes the identified configuration by considering the aforementioned components.

Configuration C_1 represents the MORGAN 's original setting, equipped with a standard NLP pipeline and the Weisfeiler–Lehman kernel. Starting from this, we derive other configurations, i.e., C_2 and C_3 by introducing the lemmatization step and a novel kernel similarity, respectively. Finally, by C_4 we combine all the newly introduced mechanisms to assess how the new tool advances compared to its preceding version by analyzing two different dimensions, i.e., the accuracy in terms of relevant results, and the delivery time. The former might benefit from the improvement of preprocessing phase as the data-driven tools strongly rely on input data to compute the outcomes. In contrast, the latter is affected by the complexity of the employed kernel function; thus, improving this component could speed up both the training and the query phases. In this respect, we expect that the Lemmatizer component increases the quality of the recommended items while the computation time might be reduced by exploiting the Vertex Histogram kernel. To conduct the experiments, we split a dataset into two independent parts, namely a training set and a testing set. In practice, the training set represents the (meta)models that have been collected ex ante, they are available at developers' disposal. The testing set represents the metamodel being developed, or *the active project*. In this way, our evaluation simulates a real-world scenario: *the system is supposed to generate recommendations for the active metamodel based on the data mined from a set of existing metamodels*. In the evaluation, we adopted the k-fold cross-validation technique widely used in evaluating ML-based applications [24]. The overall process is described in Fig. 6b, and it is applied on both the metamodel dataset and the model one presented in Sect. 4.2. 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 (see Sect. 3.2, while the latter has been split into GT graphs and query graphs. In our previous work [6], we already evaluated the performance MORGAN with different GT and query sizes to mimic the behaviors of a modeler working at different stages of a modeling project. In this paper, we resemble the situation where the metamodel under construction is almost complete, i.e., the modeler already defined the two third of classes and structural features. In particular, a single query graph represents the active context of the modeler who is defying classes and structural features for a specific model. Meanwhile, the GT graph is the elicited part extracted from the original model that should be defined by the modeler to complete the partial model. Even though this splitting strategy could lead to possible inconsistencies, we carefully encode the original models to avoid any broken references.

4.4 Metrics

Given a set of $|I|$ testing items,[23] and a partial item i, MORGAN produces a ranked list of N elements, i.e.,

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

[22] Replicating the experiments with all the configurations is out of the scope of this paper. Thus, we selected the ones that lead to a better accuracy, and used it to evaluate the novel aspect introduced in this extension.

[23] We use items to refer both to metamodels and models, for the sake of presentation.

$REC_N(i)$. We evaluate the system's performance by comparing such a list with the ground-truth data *GT(m)*. First, let us call $\text{match}(i) = |\text{GT}(i) \cap \text{REC}_N(i)|$ the number of correctly retrieved items, then we use the following three metrics: Success rate, Precision, Recall, and F-measure metrics, defined as follows [23]:

- **Success rate** measures the ratio of testing items getting at least a match in the recommendation list, to the total number of testing items:

$$\text{success rate} = \frac{\text{count}_{i \in I}(\text{match}(i) > 0)}{|I|} \tag{10}$$

- **Precision** is the ratio of number of matched elements to the number of recommended items:

$$\text{precision} = \frac{\text{match}(i)}{N} \tag{11}$$

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

$$\text{recall} = \frac{\text{match}(i)}{\text{GT}(i)} \tag{12}$$

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

$$\text{F-measure} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{13}$$

Recommendation time is a well-known delivery quality factor to evaluate recommender systems [25]. Therefore, we are interested in measuring the computational efficiency of the approach by computing the time needed to perform the whole recommendation activity, including the training and the testing phases. In particular, the two operations are defined as follows:

- **Preprocessing time** is required to encode the input data as well as extract relevant features. Since MORGAN is a data-driven tool, this phase is crucial to achieving good recommendation scores.
- **Testing time** Once being trained, the system delivers the computed recommendation given an input context, i.e., a model under construction. This operation measures to what extent MORGAN delivers relevant artifacts for a given context on average.

The time required for all the aforementioned operations is measured using a laptop with Intel i58250U CPU @ 1.60GHz, 16GB RAM, and Ubuntu 16.04 OS.

5 Experimental results

To investigate the potential contributions of the conceived extension, we replicate the study conducted in the original MORGAN paper [6] by analyzing the performances obtained with the abovementioned enhancements, i.e., lemmatization preprocessing and the Vertex Kernel strategy. The evaluation adheres to the configuration settings discussed in Sect. 4.3 to avoid any bias in the comparison. We investigate if the mechanisms presented in Sect. 3 contribute to an increase in the overall accuracy in terms of the selected metrics, i.e., success rate, precision, recall, and F-measure, as well as in a gain considering the delivery time. The experimental results are reported using violin boxplots, representing both boxplot and density traces. This aims to bring a more informative indication of the distribution's shape [26], enabling us to comprehend better the magnitude of the density. To this end, we report and analyze the experimental results by answering the three research questions introduced in Sect. 4.1. In particular, Sect. 5.1 presents the results obtained when the novel NLP preprocessing step is considered. We analyze the gain in computation time obtained by using the Vertex Kernel similarity in Sect. 5.2. The capability to supporting the completion of JSON schema is eventually discussed in Sect. 5.3.

5.1 RQ$_1$: *Does the preprocessing step contribute to a performance gain of MORGAN ?*

To assess the contribution of the lemmatizer component, we compare the former version of the tool with MORGAN equipped with the new NLP module, namely configurations C_1 and C_2, respectively, considering the two recommendation tasks related to the modeling activity, i.e., providing model classes and class members.

Recommending model classes. The comparison between the two identified configurations of MORGAN employed to support model classes recommendation is depicted in Fig. 12. It is evident that by Configuration C_2, MORGAN increases the overall accuracy even though the measured improvement is negligible in terms of the considered metrics, i.e., the distribution of the relevant items are similar to the preceding version of MORGAN . For instance, the violin boxplots representing the success rate values shown in Fig. 12a span from 0.10 to 0.60 for both the considered configurations. Despite this, by C_2, MORGAN slightly improves the relevance of the delivered items since they are more uniformly distributed than the ones retrieved by C_1. This finding is confirmed by analyzing the values obtained by the other metrics, i.e., precision, recall, and F-measure. While MORGAN suffers from some degradation of performance in terms of precision with C_2, we measure higher values for recall, meaning that the new version of the tool reduces the number of false negatives. On the one hand, the violin boxplot for C_1 shown in Fig. 12b spans from 0.0 to 0.20 while the one representing results for C_2 reaches 0.10 as a maximum value. On the other hand, Fig. 12c shows that MORGAN obtains a better recall by C_2 compared to C_1, i.e., the variance of the results is reduced by adopting the lemmatizer component. The F-measure values depicted in Fig. 12d confirm that the recommendation of model classes benefits from the enhancements, i.e., recall mitigates the lower results obtained by the precision.

Recommending class members. The improvement is more evident by analyzing the results obtained in the second modeling task, i.e., the recommendation of class members, as depicted in Fig. 13. Take as an example, the obtained success rate depicted in Fig. 13a ranges from 0.30 to 0.80 and from 0.40 to 1.00 by using C_1 and C_2, respectively. This demonstrates that the lemmatizer component is capable of increasing the relevance of the returned items as the success rate is increased by 10% on average. Such an improvement is confirmed by analyzing the results obtained for precision, recall, and F-measure shown in Fig. 13b, c, and d, respectively.

More specifically, the precision reaches the maximum value of 0.60 when C_2 is considered. Meanwhile, the former version of MORGAN represented by C_1 earns a maximum precision of 0.50 even though the distribution of the items is similar in terms of variance. Similarly, the new NLP component is capable of increasing the recall by 10% on average with respect to the former version of the tool represented by C_1's violin boxplot. Overall, adopting C_2 facilitates a better performance although the delta is minimal. This issue has been already threatened in the original work by analyzing the similarity among the models belonging to D_α.[24] We realized that the similarity of the considered models is very low, thus undermining the capability of the approach to be more effective. This is quite expected as MORGAN is a data-driven tool that strongly relies on the quality of the input data. Therefore, introducing an additional preprocessing step contributes to lead better performance even with not remarkable results.

Recommending metaclasses. Similar to the previous analysis, Fig. 14 shows the comparison of the old MORGAN , i.e., C_1 and the new one, i.e., C_2 in recommending metaclasses. It is clear that the proposed approach performs better in both recommendation contexts, i.e., all the metrics are improved by 10% on average. By considering the recommendation of metaclasses, the boxplots show that MORGAN delivers more relevant items compared to the former version, i.e., the overall variance is reduced by adopting the enhanced approach. Such an improvement becomes evident by analyzing the success rate metric shown in Fig. 14a. While using the original approach the corresponding violin plot spans from 0.20 to 0.80, the recommended metaclasses are concentrated in the upper part of the diagram by adopting the extended version of MORGAN . This means that the variance of the elements is reduced since they are distributed from 0.30 to 0.90.

Similarly, the precision, recall, and F1-measure values computed with C_2 are more equally distributed with respect to the results obtained in the previous work, represented by C_1. For instance, the distribution of the precision values presented in Fig. 14b is centered around 0.20 using C_1, implying that most of the metaclasses are not suitable for a given context. In contrast, adopting C_2 improves the quality of the retrieved artifacts, i.e., the corresponding violin span homogeneously. The same trend can be observed for the other metrics, i.e., recall and F-measure shown in Fig. 14c and d. Altogether, this means the new preprocessing component contributes to increasing the overall quality of the retrieved artifacts.

Recommending structural features. Similar to D_α, the contribution of the advanced preprocessing step is more evident when structural features are considered. In particular, the success rate values obtained by MORGAN equipped with the lemmatizer component strongly confirm our findings, since the scores are concentrated on the [0.60; 1.00] range. Meanwhile, adopting C_1, i.e., the former version of MORGAN , produces less relevant artifacts as the corresponding boxplot is scattered from 0.0 to 1.00 as shown in Fig. 15a. Roughly speaking, this means that the extended version of MORGAN comes in handy for a larger set of testing contexts, improving the overall quality of suggested structural features. The other considered metrics remark this improvement, i.e., the distribution of values obtained with MORGAN are greater than the ones of the original work, though the delta is less noticeable compared to the success rate metric. As shown in Fig. 15(b), the precision values obtained by using C_2 are very similar to those computed by using C_1, i.e., the former version of MORGAN . Nonetheless, we observe that most of the values are centered around 0.50 and 0.60 considering C_1 and C_2, respectively, meaning that the lemmatizer component improves the quality of the retrieved items even in a small percentage of cases. This finding is confirmed by the recall values as the violin plot depicted in Fig. 15(c) have almost a similar shape. Figure 15(d) eventually summarizes the obtained enhancements by showing the F-measure results for the two examined configurations. It is worth noting that C_2's violin plot reaches 0.85 as a maximum while C_1 distribution stops at 0.80. This can be explained by observing the recall boxplot since the one representing precision has the same shape. Thus, the decrease of the variance in the recommended items distribution is almost led by recall, as the F-measure score represents the harmonic mean of the two aforementioned metrics.

[24] In the scope of the paper, we do not report the similarity values among the considered models. The interested readers are kindly referred to the original version of MORGAN [6] for more detail.

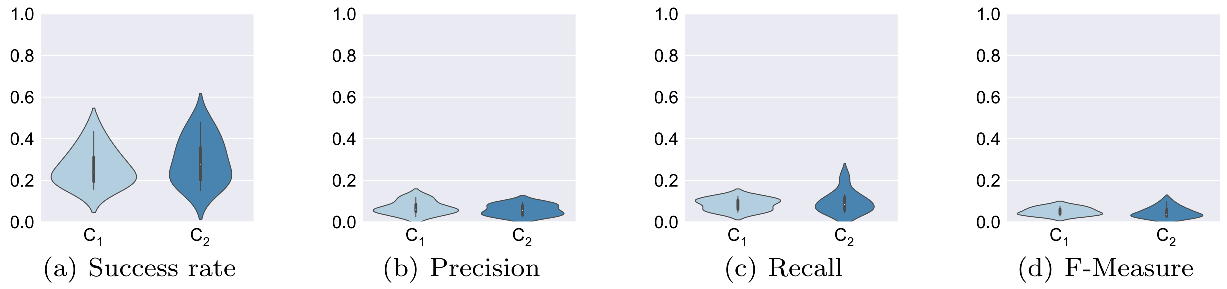


Fig. 12 Evaluation scores for recommending model classes on D_α

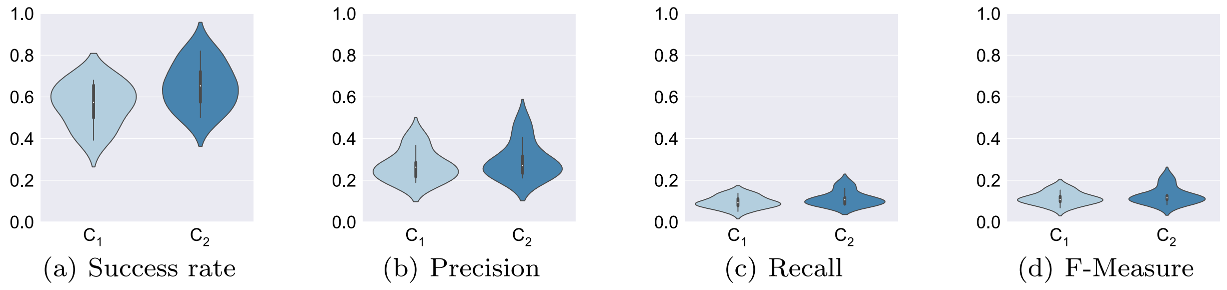


Fig. 13 Evaluation scores for recommending class members on D_α

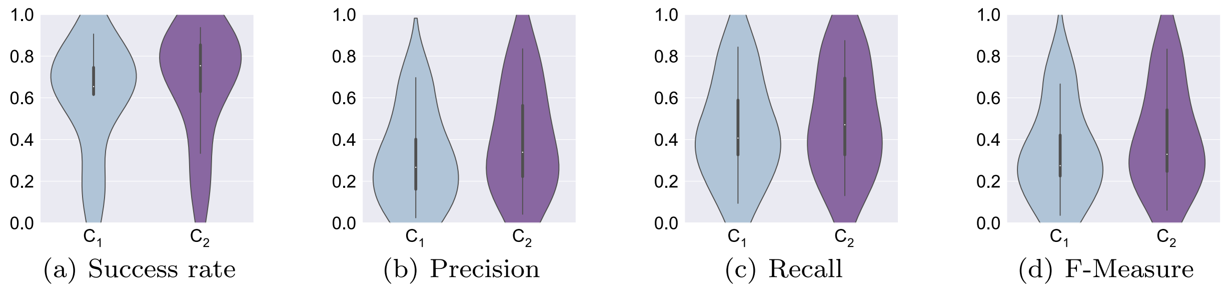


Fig. 14 Evaluation scores for recommending metaclasses D_β

To better understand the contribution of the new configuration, we compute two widely used statistical tests on the two populations, i.e., Wilcoxon rank sum test adjusted *p*-values and Cliff's *d*. Table 3 summarizes the obtained results considering the average values of the metrics for both D_α and D_β datasets. Concerning the former, we observe that the introduction of the new kernel has a negligible effect on the results as confirmed by the two statistical tests. Furthermore, the aggregated precision, recall, and F1 values are almost the same in both cases, meaning that adopting Configuration C_2 does not lead to a better performance. Such a negative result can be explained by the heterogeneity of the Java models belonging to D_α. Essentially, the introduction of the new kernel is not enough to obtain better results when D_α is considered. In contrast, we observe an improvement when MORGAN is equipped with the Vertex Histogram kernel on

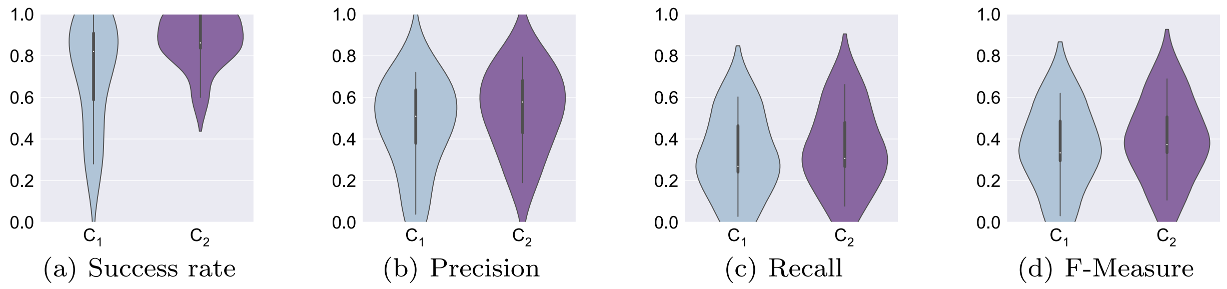


Fig. 15 Evaluation scores for recommending structural features on D_β

Table 3 Average prediction scores, Wilcoxon rank sum test adjusted *p*-values and Cliff's *d* results

	D_α				D_β			
	Class members		Classes		Structural features		Metaclasses	
	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
Avg. Success rate	0.36±0.48	0.36±0.47	0.22±0.41	0.21±0.41	0.97±0.15	0.99±0.09	0.66±0.47	0.73±0.44
Avg. Precision	0.05±0.13	0.06±0.13	0.04±0.11	0.04±0.09	0.48±0.30	0.70±0.28	0.50±0.43	0.56±0.41
Avg. Recall	0.03±0.12	0.03±0.08	0.08±0.20	0.07±0.20	0.40±0.26	0.74±0.25	0.41±0.38	0.51±0.38
Avg. F-Measure	0.03±0.10	0.03±0.07	0.04±0.10	0.04±0.09	0.57±0.23	0.86±0.22	0.39±0.40	0.54±0.42
Wilcoxon *p*-value[2]	0.6486		0.2149		0.0856		0.6812	
Cliff's *d* results[2]	−0.01295 (n)[1]		−0.0122 (n)[1]		−0.0142 (n)[1]		−0.0877 (n)[1]	

[1] magnitude values l:large, s:small, n:negligible

[2] Both Wilcoxon rank sum test adjusted *p*-values and Cliff's *d* results are computed on the success rate scores

D_β, i.e., MORGAN achieves better average success rate, precision, and recall scores than the ones achieved from the previous version for both metaclasses and structural features recommendations. Though the improvement is not statistically significant, the aggregate values show that MORGAN equipped with C_2 leads to better results by considering a more curated dataset.

Answer to RQ$_1$. Compared to the former version, MORGAN improves its performance by adopting refined preprocessing strategies, i.e., lemmatization instead of stemming. Though such improvements impact mostly on success rate, the enhanced preprocessing increases the probability of retrieving valuable artifacts.

5.2 RQ$_2$: How does the vertex histogram kernel function impact on the computational efficiency?

With the aim of assessing to what extent the time of recommendations can be improved, we compared the former version of the system with the one that exploits the Vertex Histogram kernel as a recommendation engine, i.e., considering C_1 and C_3. To this end, we analyze the computational efficiency in terms of *(i)* training time needed to learn the encoded features in the models and *(ii)* testing time, namely the time needed to get a set of recommendations given the modeler's context. Similar to the previous research question, we conducted such an evaluation by considering the two datasets of the original work, i.e., D_α and D_β. Furthermore, we investigate how the type of recommended item could affect the system from the computational point of view, i.e., if recommending metamodel classes or their structural features can lead to a different execution time. Table 4 summarizes the results of the comparison between the two aforementioned configurations when models are considered.

The measured time shows that augmenting MORGAN with Vertex Histogram results in better computational efficiency. Considering the training phase, the benefit of adopting the novel kernel is more evident for the recommendation of class members, i.e., the required time decreases from

Table 4 Timing performance on D_α (seconds)

	Classes		Class members	
Operation	C_1	C_3	C_1	C_3
Preprocessing	7570	**1242**	7815	**923**
Testing	689	**101**	868	**166**
Single rec	4.0	**0.6**	5.0	**1.0**

The numbers printed in bold correspond to the best scores

7,815 to 923 s on average. Furthermore, this operation takes 7,570 and 1,242 s for model classes when C_1 and C_3 are enabled, respectively. The Vertex Histogram contributes also to reducing the whole testing time, i.e., Configuration C_3 takes 101 and 166 s for classes and their members, respectively. Meanwhile, the former version of MORGAN that adopts C_1 requires 689 s to recommend model classes and 868 s for the considered members. Furthermore, we measure the time required to produce the recommendations for a single model. It is evident that Vertex Histogram performs better compared to the WLOA kernel, i.e., the needed time decreases from 4 to 0.6 and from 5 to 1 s for classes and members recommendations, respectively.

Concerning D_β, the results confirm that equipping MORGAN with Vertex Histogram helps speed up the overall recommendation process, i.e., MORGAN gets a better prediction when running with C_3 instead of C_1 by both considered metrics. In particular, the time required using C_1 for the training phase is 120 seconds on average for the metaclasses while adopting C_3 needs 17 seconds considering the same amount of data. Similarly, the same trend can also be seen for structural features since MORGAN equipped with the new kernel module reduces the whole training time from 153 to 51 seconds.

By the testing operation, the tool requires 14 and 9 s when C_1 and C_3 are adopted, respectively. It is worth mentioning that the computed time is the same for both metaclasses and structural features. This finding can be explained by consid-

Table 5 Timing performance on D_β(seconds)

	Metaclasses		Structural features	
Operation	C$_1$	C$_3$	C$_1$	C$_3$
Preprocessing	120	**17**	153	**51**
Testing	14.0	**9.1**	14.0	**9.1**
Single rec	1.0	**0.2**	1.5	**0.3**

The numbers printed in bold correspond to the best scores

Table 6 Comparison between C$_1$ and C$_3$ considering D_α

	Classes		Class members	
Metrics	C$_1$	C$_3$	C$_1$	C$_3$
Success rate	0.21	**0.23**	0.63	**0.64**
Precision	0.03	**0.05**	0.27	**0.29**
Recall	0.08	**0.09**	0.10	**0.11**
F-measure	0.04	**0.06**	0.11	**0.12**

The numbers printed in bold correspond to the best scores

Table 7 Comparison between C$_1$ and C$_3$ considering D_β

	Metaclasses		Structural features	
Metrics	C$_1$	C$_3$	C$_1$	C$_3$
Success rate	0.60	**0.62**	0.72	**0.78**
Precision	0.30	**0.33**	0.46	**0.49**
Recall	0.44	**0.45**	0.31	**0.32**
F-measure	0.33	**0.35**	0.34	**0.36**

The numbers printed in bold correspond to the best scores

ering the average dimension of the metamodels belonging to the $D_\beta$dataset, i.e., they include a small number of structural features of each metaclass. Therefore, the recommendation phase takes almost the same time for the two aforementioned metamodel artifacts.

This claim is confirmed by analyzing the time needed for a single recommendation, i.e., it is almost the same for a single model considering both kernels. It is evident that C_3 leads to better performances compared to the results obtained by running MORGAN with C_1. In particular, recommending a set of classes and structural features requires 0.2 and 0.3 s, respectively, by adopting C_3. Meanwhile, MORGAN equipped with the WLOA kernel takes 1.0 and 1.5 s on average for the two recommendation tasks (Table 5).

Altogether, Vertex Histogram improves the computation efficiency in all the considered scenarios, i.e., the recommendation of models and metamodels artifacts. However, MORGAN 's overall accuracy may decrease as we are employing a different technique. Therefore, we replicate the analysis conducted in the previous research question by comparing C$_1$ and C$_3$ in terms of the considered metrics, i.e., success rate, precision, recall, and F-measure. Table 6 shows the comparison by considering the average values of the mentioned metrics considering models, i.e., D_α.

It is evident that the novel graph kernel preserves the overall accuracy of the original work, i.e., all the metrics are improved on average. In particular, the introduction of Vertex Histogram leads to better results in recommending the two types of model artifacts, i.e., classes and their corresponding members. For instance, the success rate measured for model classes passes from 0.21 to 0.23 when C$_3$ is considered. Similarly, the other metrics are improved by 1% on average with respect to C$_1$. Similarly, MORGAN equipped with the new kernel strategy achieves better performance when class members are considered even though the delta is negligible. Nonetheless, Vertex Histogram aims to improve computational efficiency in the first place since better performance in terms of observed metrics is obtained by means of the new preprocessing component.

Table 7 confirms that MORGAN 's prediction scores are not hampered by the introduction of the new graph kernel. The table shows that the examined metrics are improved up to 2% apart from the success rate measured for structural features, i.e., its score reaches 0.78 using C$_3$ while using C$_1$ yields 0.72 as the maximum. Such an improvement can be explained by considering the strong similarity among the considered metamodels. Altogether, the observed results demonstrate that the contribution of the Vertex Histogram nurtures better results with respect to accuracy alongside the time required for the whole recommendation process.

Answer to RQ$_2$. Equipping MORGAN with the Vertex Histogram kernel helps improve the computation efficiency. Moreover, the overall prediction accuracy has also been slightly enhanced, meaning that the kernel strategy contributes to a performance gain, albeit marginal.

5.3 RQ$_3$: *How effective is MORGAN at recommending JSON schema elements?*

To examine the generalizability of the tool, we assess MORGAN 's capability of supporting the two modeling completion tasks over the $D_\gamma$dataset composed of JSON schema, namely root properties and the nested ones that can be mapped to metaclasses and structural features, respectively, as discussed in Sect. 2.1. Thus, we conduct the same evaluation presented in the previous subsections by using the configuration that embodies the novel components presented in this extension, i.e., Configuration C$_4$, that includes the lemmatizer component and the Vertex Histogram kernel. *Recommending JSON root properties.* Fig. 16 shows the results obtained when MORGAN is employed to recommend JSON schema properties given an incomplete model. The success rate scores span from 0.2 to 1.00. This means that MORGAN recommends at least one correct property in almost all of the examined contexts. In contrast, we experience some performance degradation on the other metrics. For instance, the precision boxplot ranges from 0.0 to 0.40, i.e., relevant properties are recommended with a low probability on average. In this respect, these results are similar to the ones obtained for the models belonging to D_α. This finding is confirmed by the recall values as the number of items properly delivered is higher, i.e., the maximum value reaches 0.60. The distribution of the F-measure values resembles the precision ones, suggesting that false positives have a negative effect on the overall performances. By carefully inspecting the results, we observe that the D_γ dataset is very heterogeneous since all the schema have been extracted from GitHub repositories. Thus, such degradation of performance is due to the nature of the data even though the novelties introduced in the approach mitigates this issue.

Recommending JSON nested properties. Fig. 17 summarizes the results obtained by MORGAN in recommending nested JSON properties. Similar to the two examined datasets in Sect. 5.1, the tool obtains better results compared to the root properties recommendations. For instance, the success rate achieves 0.70 on average as we can observe from the corresponding boxplot. Furthermore, the majority of the values span from 0.60 to 0.90, meaning that nested properties have been recommended in most of the cases. A similar trend can be observed by analyzing the precision violin plot even though the average value is around 0.60. However, MORGAN suffers from some degradation in performance in terms of recall, i.e., the corresponding violin plot spans from 0 to 0.80, with an average value of 0.30. In other words, the system fails to detect false negatives when JSON nested properties are considered. This impacts negatively on the F-measure metric as the corresponding violin plot has a similar shape compared to the recall one. Despite this, the conducted study aims to demonstrate the capability of MORGAN in recommending different modeling artifacts other than Ecore models and class diagrams. It is our firm belief that improving the quality of the considered JSON schema will lead to a better accuracy. For instance, we can enhance the feature extraction process to include more relevant data embedded in a JSON schema, e.g., the type of the properties.

At its current status, the system can provide (i) attributes and relationships to enrich a class or (ii) a list of similar classes considering the corresponding structural features. In principle, we could adapt the conceived parser module to extract relevant features from any kind of modeling artifacts. Therefore, MORGAN can possibly support the completion of a state machine model if being properly trained with models, i.e., a dataset composed of state machine models with a decent number of transitions. However, the final accuracy depends a lot on the quality of training data. Altogether, completion of state machine models is possible under certain conditions, i.e., the availability of a proper dataset, and the refactoring of the parser component. This, however, needs to be validated with real-world datasets, and we consider it as our future work.

Table 8 Evaluation scores for D_δ

Metric	Metaclasses	Structural features
Success rate	0.62	0.59
Precision	0.34	0.15
Recall	0.45	0.28
F-Measure	0.31	0.15
Time (s)	0.04	0.03

Answer to RQ$_3$. MORGAN succeeds in supporting JSON schema completion, though we experienced some degradation in performance for some metrics, i.e., the recall scores are very low. Such negative results might be mitigated by improving the quality of the input schemas.

5.4 RQ$_4$: *How is MORGAN 's performance changed when working on ModelSet, a benchmark dataset of metamodels and UML models?*

To further study the MORGAN 's performance, we extracted two different datasets from ModelSet, namely D_δ and D_ϵ, by using the provided API as described in Sect. 4.2. Table 8 summarizes the results obtained by MORGAN on D_δ. Similar to RQ$_3$, we conducted the experiment for this research question using C$_4$. It is worth noting that the results are similar to those obtained with D_α, meaning that the degree of similarity among the artifacts is almost the same. Concerning D_δ, MORGAN obtains better performance in recommending metaclasses, i.e., all the scores are higher compared to the ones obtained for the structural features. In particular, the recall score is 0.45 in recommending metaclasses, while it is 0.28 for structural features. In other words, the number of false negatives is fewer when metaclasses are considered. Concerning the time for a obtaining a single recommendation, MORGAN is very fast, i.e., retrieving the suggested item requires only 0.04 and 0.03 s for metaclasses and structural features, respectively. Nevertheless, the required time strongly depends on the size of the considered artifacts, thus leading to some scalability issues if a larger dataset is considered. In fact, MORGAN is faster on D_δ compared to D_α because the considered metamodels are smaller in terms of the number of attributes and relationships.

A similar trend can be observed for D_ϵ, i.e., recommending UML classes leads to better performance as shown in Table 9. Even though the success rate is slightly lower, i.e., 0.58 and 0.62 for classes and class members, respectively, the other metrics confirm that MORGAN obtains better results in recommending class entities compared to the former model dataset, i.e., D_α. This is expected since we extracted those

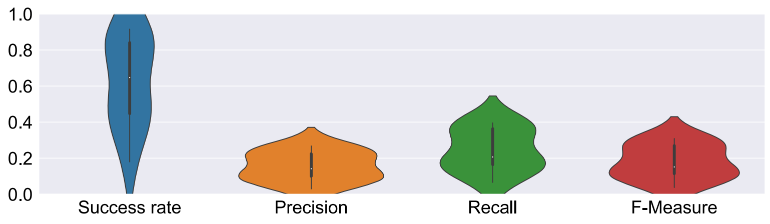


Fig. 16 Evaluation scores for recommending JSON root properties on Dγ

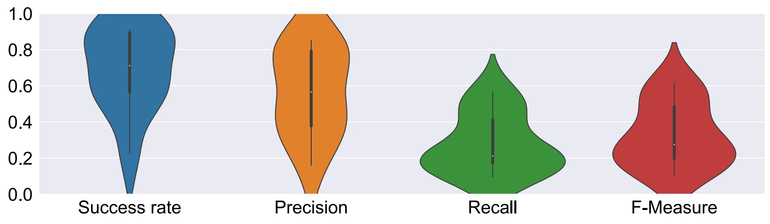


Fig. 17 Evaluation scores for recommending JSON nested properties on Dγ

Table 9 Evaluation scores for D_ϵ

Metric	Classes	Class members
Success rate	0.56	0.61
Precision	0.24	0.15
Recall	0.33	0.19
F-measure	0.20	0.13
Time (s)	0.21	0.13

models by using MoDisco from Java projects while $D_\epsilon$is composed of a curated list of UML models. Despite this, the scalability issue is confirmed by observing the required time for obtaining a single recommendation, i.e., MORGAN takes 0.20 s to suggest a list of relevant UML classes. Though the time required for the class attributes is slightly lower, i.e., 0.13 s, it is evident that augmenting the complexity of the graphs can hamper the tool's timing performance. Altogether, the conducted analysis suggests that the dataset curation process contributes to achieving better results even though the scalability of the approach can be affected.

Answer to RQ$_4$. The obtained results obtained by considering ModelSetare comparable with the ones reported in the previous analysis, though MORGAN suffers from the scalability issue if complex graphs are considered.

6 Threats to validity

We discuss the threats that could impact on the validity of the study's outcomes. Moreover, we also identify possible countermeasures to mitigate them.

Threats that arose in the former work *internal validity* were related to two aspects, i.e., the graph kernel similarity and the employed encoding scheme, including the preprocessing of the input models. Concerning the former, we enhance the underpinning kernel similarity by adopting the Vertex Histogram technique. In such a way, we increase the computational efficiency by reducing the whole recommendation time even for large graphs. Concerning the latter, the pre-processing phase might miss relevant data, i.e., stemming does not consider the semantics of the examined terms, leading to possible loss of features during the encoding phase. To minimize the threat, we employed lemmatization in the pre-processing pipeline to augment the overall accuracy of the tool. Moreover, the conducted evaluation on the ModelSetdatasets reveals that the scalability of MORGAN can be affected when increasing the size of the training data. To mitigate this threat, further study on the underpinning algorithm is needed, and we leave this as future work.

The selection of JSON schema as modeling activity may hamper the *external validity* of our findings. Though they conform to a well-defined metamodel, the internal structure is completely different from that of the artifacts examined in the original work. To tackle this issue, we adapted MORGAN 's encoder component by introducing a tailored parser for JSON schema to obtain the same format used for the other artifacts, i.e., models and metamodels. Furthermore, the results in terms of accuracy might be undermined by the quality of the considered schemas, i.e., the similarity among the JSON belonging to the dataset. We mitigate this threat by applying a set of quality filters on the JSON schemas crawled from GitHub, i.e., duplicates removal, parsing, and conformance check.

7 Related work

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

7.1 Modeling assistant tools

The Extremo Eclipse plugin [27] 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.

Dupont et al. [28] propose a domain-specific modeling (DSM) environment to assist the specification of models in Papyrus. Given a UML profile, the tool exploits the EMF generator to map each profile metaclass to a concrete Java class. Afterward, the generated Papyrus plugin is used to define custom DSL using a tooling palette to specify graphic components. The proposed environment can be further extended by including different features, e.g., proactive triggering, fine-grain palette customizations, or suggestions to complete the input UML profile, to list a few.

AVIDA-MDE [29] supports the generation of behavioral models starting from the requirements specification by using a digital evolution strategy. Given a well-defined set of parameters and an initial model, the tool elicits instinctual knowledge elements and constructs new transitions to support scenarios and meet the specified constraints. The resultant model is eventually evaluated using a robot navigation system as the testing scenario.

With the aim of supporting the definition of Atom^3 models, Sen et al. [30] devised a model assistant based on a constraint logic program (CLP) strategy. The proposed approach induces the complete model using a set of constraints expressed in Prolog. Afterward, the user can customize such generated model by relying on a domain-specific editor. The system eventually solves the specified constraints to produce the final model, which can be decorated with additional elements manually specified by the designer.

The ASketch tool [31] is capable of completing Alloy partial models using automated analysis. Given the input model, the interpreter extracts relevant information using a tailored parser. In such a way, a list of possible abstract candidates is generated and ASketch can find possible solutions by employing an SAT solver. The system eventually fills the model under construction with concrete candidate fragments such that predefined tests, i.e., unit testing, test execution.

Batot and Sahraoui [32] 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. An interactive approach to metamodel construction has been proposed in Ref. [33]. Given a model fragment expressed using graphical tools, the system can suggest relevant elements that can be employed to complete the model-under-construction by using well-known refactoring strategies, quality issues for the conceptual scheme, model design patterns, and antipatterns. The system assesses the quality of this initial version against model examples. The validated model is eventually compiled into a given technology, ie EMF or METADEPTH. Similarly, Kuschke et al. [34] 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.

Given an incomplete Simulink model, the SimVMA prototype [35] integrates ML, MDE, and software cloning to provide (i) a complete model; or (ii) single edit operation on the incomplete model. To this end, the tool employs code cloning techniques and an ML model to identify potential similar candidates by examining past usage statistics of existing models.

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 [36] 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.

In the context of metamodeling assistant, we already proposed MemoRec [22], a recommender system that exploits a context-aware collaborative filtering technique to retrieve relevant artifacts. By relying on four different encodings, the system assists the completion of a given metamodel by suggesting metaclasses at the level of the package and the corresponding structural features. After the encoding phase, MemoRec retrieves a list of items according to the type of the selected recommendation ranked by making use of the Jaccard distance similarity.

Compared with the reviewed studies, MORGAN is different as it supports various types of models expressed in the eligible formats, i.e., *Ecore*, *xmi*, and JSON schema. Moreover, using even partial models in the training phase allows MORGAN to obtain acceptable results.

7.2 Using graph kernels to develop recommender systems

The usage of random walk kernels has been investigated to support social network-based recommendations [37]. Once the social influence has been encoded in the graph, the employed kernel predicts the ratings of the corresponding products by using a distance measure defined on stationary transition probabilities.

Ostuni et al. [38] exploits a neighborhood-based graph kernel to improve content-based recommender systems. By considering well-founded Linked Open Data datasets, the proposed approach extracts semantics from stored items and encodes their description into graphs. Afterward, the proposed h-hop item neighborhood algorithm is employed to retrieve the most similar items by relying on graph kernel similarity that exploits the occurrence and the locality of the entities.

An empirical evaluation of several graph kernel strategies including random-walk with-restart similarity matrix and regularized commute-time kernels was conducted [39] to study their performance. The authors applied these techniques to perform (i) semisupervised classification and (ii) collaborative recommendations. To assess the effectiveness of the considered approaches, a nested cross-fold validation has been carried on by measuring the AUC curve for each algorithm.

A generic kernel-based machine learning approach of link prediction in bipartite graphs has been proposed [40] to support collaborative filtering recommender systems. First, the system inspects nodes and links that are close to a focal user-item pair to predict the possible interactions. Then, the graph kernel is used to capture the structure of the considered context to feed the SVM algorithm that is employed to deliver the retrieved items.

Xu et al. [41] made use of graph kernels to compute item similarity by exploiting user-item ratings. First, the approach builds an undirected graph from the user-item matrix. Afterward, such graphs are transformed into neighborhood graphs that have been used to compute the graph kernel similarly. The results show that the proposed outperforms traditional recommender systems based on linked open data strategy.

In relation to existing work, MORGAN is the first multi-purpose recommender system to support the specification of different modeling artifacts, including JSON schema. Being built on top of graph kernels allows the tool to exploit the well-founded background related to such techniques to deal with data presented as nodes and edges. In this respect, there is the potential to further improve MORGAN 's performance by incorporating advanced algorithms in this domain. We consider this as our future research agenda.

8 Conclusion and future work

To support modelers in their daily activities, MORGAN has been proposed as a practical solution to the recommendation of models and metamodels, allowing one to easily choose and adopt suitable components. In this paper, we extended our previous work by introducing the lemmatization step in the preprocessing phase. Furthermore, we equipped the recommendation engine with a more efficient kernel similarity function, helping the system to obtain more relevant results in less time. An empirical evaluation of five real-world datasets demonstrated that MORGAN is applicable in different application domains.

For future work, we plan to further improve MORGAN by adopting different graph structures, e.g., heterogeneous or weighted graphs, to improve the scalability of the approach. Furthermore, we suppose that link prediction or generative graphs techniques can be applied as an alternative strategy to complete models represented in a graph-based format. Concerning the recommendation of JSON schema elements, it is our strong belief that a curated dataset with similar elements brings better results in terms of accuracy. Last but not least, we will investigate the applicability of ontologies to increase the number of relevant artifacts, moving forward a domain-aware intelligent modeling assistant capable of embedding the semantics in the retrieved recommendations.

Acknowledgements The research described in this paper has been partially supported by the AIDOaRT Project, which has received funding from the European Union's H2020-ECSEL-2020, Federal Ministry of Education, Science and Research, Grant Agreement n° 101007350. We thank the anonymous reviewers for their valuable comments and suggestions that helped us improve the paper.

Funding Open access funding provided by Università degli Studi dell'Aquila within the CRUI-CARE Agreement.

References

1. Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., Iovino, L.: Automated classification of metamodel repositories: a machine learning approach. In: 2019 ACM/IEEE 22nd International Con-

ference on Model Driven Engineering Languages and Systems (MODELS), pp. 272–282, Sep 2019
2. Nguyen, P.T., Ruscio, D.D., Pierantonio, A., Rocco, J.D., Iovino, L.: Convolutional neural networks for enhanced classification mechanisms of metamodels. J. Syst. Softw. **172**, 110860 (2021)
3. Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., Búr, M., Burgueño, L., Engels, G., Jeanjean, P., Jézéquel, J.-M., Kühn, T., Mosser, S., Sahraoui, H., Syriani, E., Varró, D., Weyssow, M.: Opportunities in intelligent modeling assistance. Softw. Syst. Model. **19**(5), 1045–1053 (2020)
4. Burgueño, L., Clarisó, R., Gérard, S., Li, S., Cabot, J.: An nlp-based architecture for the autocompletion of partial domain models. In: M. L. Rosa, S. W. Sadiq, and E. Teniente (eds.), Advanced Information Systems Engineering - 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28 - July 2, 2021, Proceedings, vol. 12751, pp. 91–106. Springer, Heidelberg (2021)
5. Weyssow, M., Sahraoui, H.A., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Softw. Syst. Model. **21**(3), 1071–1089 (2022)
6. Di Rocco, J., Di Sipio, C., Di Ruscio, D., Nguyen, P. T.: A gnn-based recommender system to assist the specification of metamodels and models. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 70–81, (2021)
7. JSON schema. http://json-schema.org/. Accessed 29 Feb 2022
8. Colantoni, A., Garmendia, A., Berardinelli, L., Wimmer, M., Bräuer, J.: Leveraging model-driven technologies for json artefacts: The shipyard case study. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 250–260, (2021)
9. Sugiyama, M., Borgwardt, K. M.: Halting in random walk kernels. In: Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1, NIPS'15, pp. 1639-1647, MIT Press, Cambridge, MA, USA, (2015)
10. López, J.A.H., Cánovas Izquierdo, J.L., Cuadrado, J.S.: Modelset: a dataset for machine learning in model-driven engineering. Softw. Syst. Model. **21**(3), 967–986 (2022)
11. Riesen, K., Bunke, H.: Graph classification and clustering based on vector space embedding. World Scientific Publishing Co. Inc., USA (2010)
12. Vishwanathan, S., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. J. Mach. Learn. Res. **11**(40), 1201–1242 (2010)
13. Clarisó, R., Cabot, J.: Applying graph kernels to model-driven engineering problems. In: Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, MASES 2018, pp. 1–5, Association for Computing Machinery, New York, NY, USA (2018)
14. Kriege, N. M., Giscard, P.-L., Wilson, R.: On Valid Optimal Assignment Kernels and Applications to Graph Classification. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016
15. Weisfeiler, B., Leman, A.: The reduction of a graph to canonical form and the algebra which appears therein. NTI Ser. **2**(9), 12–16 (1968)
16. Porter, M.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)
17. Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., Vazirgiannis, M.: GraKeL: A graph kernel library in Python. arXiv:1806.02193 [cs, stat], Mar 2020
18. Babur, Ö.: A labeled Ecore metamodel dataset for domain clustering, (2019)
19. GitHub. https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting. Accessed 29 Jan 2021
20. GitHub Archive Dataset. https://console.cloud.google.com/marketplace/product/github/github-repos. Accessed 29 Jan 2021
21. López, J. A. H., Cuadrado, J. S.: Mar: a structure-based search engine for models. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '20, pp. 57–67, Association for Computing Machinery, New York, NY, USA (2020)
22. Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P.T., Pierantonio, A.: Memorec: a recommender system for assisting modelers in specifying metamodels. Softw. Syst. Model. **2022**, 1–21 (2022)
23. Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., Di Penta, M.: FOCUS: a recommender system for mining API function calls and usage patterns. In: Atlee, J. M., Bultan, T., and Whittle, J. (eds.) Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 1050–1060. IEEE / ACM, (2019)
24. Raschka, S.: Model evaluation, model selection, and algorithm selection in machine learning. *CoRR*, abs/1811.12808, 2018
25. Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T. (eds.): Springer, Berlin, Heidelberg (2014)
26. Hintze, J.L., Nelson, R.D.: Violin plots: a box plot-density trace synergism. Am. Stat. **52**(2), 181–184 (1998)
27. Mora Segura, A., de Lara, J.: Extremo: an Eclipse plugin for modelling and meta-modelling assistance. Sci. Comput. Program. **180**, 71–80 (2019)
28. Dupont, G., Mustafiz, S., Khendek, F., Toeroe, M.: Building Domain-Specific Modelling Environments with Papyrus: An Experience Report. In 2018 IEEE/ACM 10th International Workshop on Modelling in Software Engineering (MiSE), pp. 49–56, May 2018. ISSN: 2575-4475
29. Goldsby, H. J., Cheng, B. H.: Avida-MDE: a digital evolution approach to generating models of adaptive software behavior. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation—GECCO '08, p. 1751, ACM Press, Atlanta, GA, USA, (2008)
30. Sen, S., Baudry, B., Vangheluwe, H.: Towards domain-specific model editors with automatic model completion. Simulation **86**(2), 109–126 (2010)
31. Wang, K., Sullivan, A., Marinov, D., Khurshid, S.: Asketch: a sketching framework for alloy. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2018)
32. Batot, E., Sahraoui, H.: 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, pp. 374–384, ACM, Saint-malo France, Oct 2016
33. López-Fernández, J.J., Cuadrado, J.S., Guerra, E., de Lara, J.: Example-driven meta-model development. Softw. Syst. Model. **14**(4), 1323–1347 (2015)
34. Kuschke, T., Mäder, P., Rempel, P.: Recommending Auto-completions for Software Modeling Activities. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., and Clarke, P. (eds.) Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science, pp. 170–186, Springer, Berlin, Heidelberg (2013)
35. Stephan, M.: Towards a cognizant virtual software modeling assistant using model clones. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 21–24, May 2019
36. Saini, R., Mussbacher, G., Guo, J. L. C., Kienzle, J.: 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, MODELS '20, Association for Computing Machinery, New York, NY, USA, (2020)

37. Li, X., Su, X., Wang, M.: Social network-based recommendation: a graph random walk kernel approach. In: Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL '12, pp. 409–410, Association for Computing Machinery, New York, NY, USA (2012)
38. Ostuni, V. C., Noia, T. D., Mirizzi, R., Sciascio, E. D.: A linked data recommender system using a neighborhood-based graph kernel. In *International conference on electronic commerce and web technologies*, pp. 89–100. Springer, Heidelberg (2014)
39. Fouss, F., Francoisse, K., Yen, L., Pirotte, A., Saerens, M.: An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. Neural Netw. **31**, 53–72 (2012)
40. Li, X., Chen, H.: Recommendation as link prediction in bipartite graphs: a graph kernel-based machine learning approach. Decis. Support Syst. **54**(2), 880–890 (2013)
41. Xu, W., Xu, Z., Zhao, B.: A graph kernel based item similarity measure for top-n recommendation. In: Web Information Systems and Applications: 16th International Conference, WISA 2019, Qingdao, China, September 20-22, 2019, Proceedings, pp. 684–689, Springer, Berlin, Heidelberg (2019)

Claudio Di Sipio is a PhD student at the University of L'Aquila, Italy. He is working on mining techniques to analyze open-source software, and he is also investigating the application of low-code platforms to support the development of recommendation systems. Contact him at claudio.disipio@univaq.it

Juri Di Rocco is an assistant professor at the University of L'Aquila, Italy. He obtained a PhD in Computer Science from the University of L'Aquila. He is interested in several aspects of software language engineering and Model-Driven Engineering (MDE) including domain specific modelling languages, model transformation, model differencing, modelling repositories and mining techniques. More information is available at http://www.di.univaq.it/juri.dirocco. Contact him at juri.dirocco@univaq.it.

Davide Di Ruscio is an Associate Professor at the DISIM—University of L'Aquila. His main research interests are related to several aspects of Software Engineering, Open-Source Software, and Model-Driven Engineering (MDE) including domain specific modelling languages, model transformation, model differencing, coupled evolution, and recommendation systems. He has published more than 170 papers in various journals, conferences and workshops on such topics. He is a member of the steering committee of the International Conference on Model Transformation (ICMT), of the Software Language Engineering (SLE) conference, of the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATTOSE), of the Workshop on Modelling in Software Engineering at ICSE (MiSE) and of the International Workshop on Robotics Software Engineering (RoSE). He is in the editorial board of the International Journal on Software and Systems Modeling (SoSyM), of IEEE Software, of the Journal of Object Technology, and of the IET Software journal. More information is available at http://people.disim.univaq.it/diruscio/. Contact him at davide.diruscio@univaq.it.

Phuong T. Nguyen obtained a PhD in Computer Science from the University of Jena, Germany. Since graduation, he has worked as a university teaching and research assistant in Vietnam and Italy. He is now with the University of L'Aquila, Italy, as a tenure track assistant professor. His research interests include Computer Networks, Semantic Web, Recommender Systems, and Machine Learning. Recently, he has been working to develop recommender systems in Software Engineering for mining open-source code repositories. Email: phuong.nguyen@univaq.it.