JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Integrating the Support for Machine Learning of Inter-Model Relations in Model Views

James Pontes Miranda*, **Hugo Bruneliere***, **Massimo Tisi***, **and Gerson Sunyé**†

*IMT Atlantique & LS2N (CNRS), France
†Nantes University & LS2N (CNRS), France

ABSTRACT Model-driven engineering (MDE) supports the engineering of complex systems via multiple models representing various aspects of the system. These interrelated models are usually heterogeneous and specified using complementary modeling languages. Thus, model-view solutions can be employed to federate these models more transparently. Inter-model links in model views can sometimes be automatically computed via explicitly written matching rules. However, in some cases, matching rules would be too complex (or even impossible) to write, but inter-model links may be inferred by analyzing previous examples instead. In this paper, we propose a Machine Learning (ML)-backed approach for expressing and computing such model views. Notably, we aim at making the use of ML in this context as simple as possible. To this end, we refined and extended the ViewPoint Definition Language (VPDL) from the EMF Views model-view solution to integrate the use of dedicated Heterogeneous Graph Neural Networks (HGNNs). These view-specific HGNNs are trained with appropriate sets of contributing models before being used for inferring links to be added to the views. We validated our approach by implementing a prototype combining EMF Views with PyEcore and PyTorch Geometric. Our experiments show promising results regarding the ease-of-use of our approach and the relevance of the inferred inter-model links.

KEYWORDS MDE, Modeling languages, Model Views, Machine Learning, Graph Neural Networks.

1. Introduction

Complex systems engineering is challenging, notably because of the various aspects to be tackled and information fragmentation among stakeholders (Bruneliere et al. 2022; Afzal et al. 2018). To overcome this, Model-Driven Engineering (MDE) fosters using multiple models as fundamental artifacts to support analysts, engineers, etc., in performing their tasks more efficiently. However, dealing with heterogeneous models defined in different modeling languages at various abstraction levels (e. g., Unified Modeling Language - UML, System Modeling Language - SysML, Business Process Modeling Notation - BPMN, or DSLs for problem-specific tasks) requires appropriate model federation strategies.

JOT reference format:
James Pontes Miranda, Hugo Bruneliere, Massimo Tisi, and Gerson Sunyé. *Integrating the Support for Machine Learning of Inter-Model Relations in Model Views*. Journal of Object Technology. Vol. 23, No. 3, 2024.
http://dx.doi.org/10.5381/jot.2024.23.3.a4

Model View solutions are efficient ways to deal with model federation (Bruneliere et al. 2019). Model views are built over one or several existing models, called contributing models, that potentially conform to different metamodels. A model view allows users to access information from different contributing models in an integrated and transparent way. The elements of the contributing models are integrated by adding new *inter-model links* to the model view. Such links are the concrete instances of an *inter-model relation* among the contributing models. Sometimes, inter-model links can be automatically computed via matching rules written by the engineers. However, specifying such rules requires a deep knowledge of the involved models and metamodels. Moreover, these rules may be too complex to write manually using a query language. Consequently, automating the derivation of these rules has already been identified as an important challenge (Bucchiarone et al. 2020).

Machine Learning (ML) approaches have already been exploited to improve various model management operations (De-

hghani et al. 2022; Tang et al. 2019; Barriga, Heldal, et al. 2022; Weyssow et al. 2022). For example, Graph Neural Networks (GNNs), i. e., deep-learning models optimized for operation on graph-structured data, have already been successfully used in recommendation systems for model editing (Di Rocco et al. 2021) or for the generation of structurally-realistic models (López & Cuadrado 2023). Among other benefits, they have notably demonstrated their ability to capture interesting structural properties of model graphs from a limited set of examples.

In this paper[1], we propose an ML-backed approach for computing model views that require the inference of inter-model links. Our objective is to simplify the view engineer's work by simplifying the use of ML as much as possible so that she/he does not need to write ML code. To realize our approach, we rely on the link prediction capabilities of GNNs. Unlike previous research efforts, we rely on Heterogeneous Graph Neural Networks (HGNNs), a particular class of GNNs with native support for graphs whose nodes and edges have different types (which is the case in model views). As a result, we extended the EMF Views solution (Bruneliere et al. 2020) and its ViewPoint Definition Language (VPDL) to integrate HGNNs. The engineer only needs to indicate 1) the relations to learn, 2) the parts of the models involved in the learning process, and 3) a relevant set of sample links. A declarative description of the architecture and configuration for the corresponding HGNNs is then automatically generated and can be manually updated. These HGNNs are transparently trained and finally used to infer inter-model links integrated into model views. We built a prototype of our approach and applied it to two sample case studies. We measured a promising accuracy in the inference of inter-model links.

The remainder of this paper is structured as follows. Section 2 introduces the main background of our work and motivates it via a running example. Then, Section 3 presents the proposed approach, and Section 4 describes its current implementation. Section 5 explains the evaluation we have performed at this stage. Finally, Section 6 discusses the related work before Section 7 concludes the paper by opening on the next steps of our work.

2. Background and Motivation

2.1. Model Views

This section introduces the main concepts of model views (Bruneliere et al. 2019). Our work focuses on the EMF Views model-view solution (Bruneliere et al. 2020, 2015) that uses the VPDL language to specify viewpoints and build corresponding views.

As shown in Figure 1, different aspects of a given system are represented by different models that conform to various metamodels (i. e., modeling languages). At the metamodel-level, a viewpoint is specified to determine which concepts and properties from the contributing metamodels should appear or not in corresponding views, and how these concepts should be interrelated. At the model-level, a view is then built to federate the different contributing models according to this viewpoint specification. EMF Views materializes viewpoints and views as virtual metamodels and models (respectively). A virtual (meta)model is a particular type of (meta)model that points to elements coming from the (meta)models it is based on. Thus, there is no model element duplication: only the additional information, notably the inter-model links, is stored in the view.

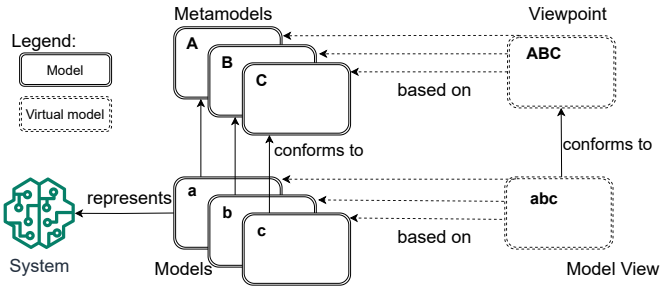

Figure 1 Main concepts of model views

The selection of which concepts and properties from the contributing metamodels must be included in the viewpoint is often simple. However, the systematic definition of how to relate concepts coming from different metamodels can be more challenging. Up to our current knowledge (Bruneliere et al. 2019), the related work focuses on the manual definition of rules for inter-model links, i. e., matching rules. We intend to address scenarios where the engineers are not establishing matching rules, but rather inferring inter-model links from previous examples. To this end, we propose to rely on Graph Neural Networks (GNNs), presented in the next section.

2.2. Graph Neural Networks

GNNs are a class of deep learning models that can operate on graph-structured data intending to learn structural and semantic graph patterns (Scarselli et al. 2009; Kipf & Welling 2017). Formally, being a graph $G = (V, E)$, a set of nodes V connected by edges in the set E, each node $v \in V$ and edge $e \in E$, they can be associated with some mapping functions $\phi(v) : V \to A$ and $\varphi(e) : E \to R$, where A and R denote the sets of node types and edge types, respectively. In homogeneous graphs, $|A| = |R| = 1$, i. e., all nodes (and edges) have the same type. In heterogeneous graph $|A| + |R| > 2$, i. e., there are different types of nodes and/or edges (Shi et al. 2022a).

2.2.1. Heterogeneous Graph Neural Networks

Heterogeneous Graph Neural Networks (HGNNs) are neural networks operating on heterogeneous graphs. An HGNN learns node representations that capture the structural and semantic information of the graph, considering the heterogeneity of nodes and edges. As standard GNNs, these models learn informative node representations through message passing and aggregation mechanisms. In each message-passing step, information is exchanged between nodes and their neighboring nodes based on the types of edges connecting them. As in other deep-learning algorithms, node embeddings must be set up as a numerical representation of its features.

[1] This journal paper extends the short paper published at the 39th ACM/SIGAPP Symposium on Applied Computing (SAC 2024) (Pontes Miranda et al. 2024).

2.2.2. Link Prediction

(H)GNNs as inference machines can be used for various downstream tasks, e. g. node predictions, graph predictions, and link predictions. In this paper, we focus on link prediction. Given a graph G, the link prediction task can be defined as computing the likelihood of observing a link in G between any two nodes v_x and v_y in V. Various techniques have been proposed to tackle link prediction, e. g., for similarity scores between nodes in social networks (Liben-Nowell & Kleinberg 2003; Perozzi et al. 2014) or recommendation systems (Lü et al. 2012). Being those above v_x and v_y in V, the likelihood of a link between v_x and v_y can be given by a parameterized function $f_\theta(v_x, v_y)$. Being the embeddings u_{v_x} and u_{v_y}, f_θ can operate directly with these numerical vectors, so $f_\theta(v_x, v_y) \approx f_\theta(u_{v_x}, u_{v_y})$. The role of the HGNNs in this scenario is to apply the message passing and aggregation layers on the embedding vectors to capture more complicated information based on node and edge types (Shi et al. 2022b).

We argue that the inference of inter-model links between contributing models of a view can be reduced to a link prediction task. In this paper, we show that HGNNs can be used to learn a joint representation of the models and infer their links.

2.3. Running Example

We now introduce a running example that we use throughout the paper to motivate our work and illustrate our proposed approach. Since our integration of HGNNs with model views is domain-independent, we have deliberately chosen not to present a complex engineering example. We rather consider two simple but significantly large models with real-world inter-model links that we can actually use for training.

In our example, a Users model contains personal information on users that can be extracted from a social network. A Movies model contains information about movies that can be extracted from a film database. We want to automatically compute a model view containing users, movies, and links connecting each user with the *movies they probably watched*.

If we were building this view without any information on the movies actually watched by our users, we would embed in the view some logical formula to estimate these links. For instance we could suppose that every user watched all movies tagged with the user's occupation (because we are all interested in movies that involve our job!). Of course such estimation formula may be arbitrarily complex (supposing that the viewpoint definition language contains a Turing complete expression language) and may take into account any information in the contributing models (i. e., the user profiles and movie database).

Let us instead suppose that we have also another large dataset, similarly structured, describing *other users*, but including also information about the movies actually watched by each users. Now we can try to automatically exploit this historical dataset, to provide a better estimation of the inter-model links between our users and movies.

In practice, from the historical dataset, we want to automatically learn a mathematical relation between each user and the movies she/he watched. Then we want this relation to be automatically applied in the view we are building to compute new inter-model links between our users and the movies they probably watched. Note that in this paper we are not interested in an explanation of the logic of the learnt relation, but only on obtaining accurate inter-model links for all users. For this, we make use of HGNNs, integrated in the viewpoint definition language.

In our experimentation, we build our models using data from the well-known *MovieLens* dataset (Harper & Konstan 2016). Later we show that the estimation we obtain in this case study has good accuracy. However, the focus of the paper is not on obtaining a good estimation for links, as this is strongly dependent on the considered use case, the quality of the dataset, and the topology and parametrization of the HGNNs. Our contribution lies in the integration of HGNNs in the view definition language, aiming at increasing the usability of this technology by non-experts in machine learning. In particular, view engineers do not have to write any Python code, but only declarative specifications, to execute the training and inference of the HGNN.

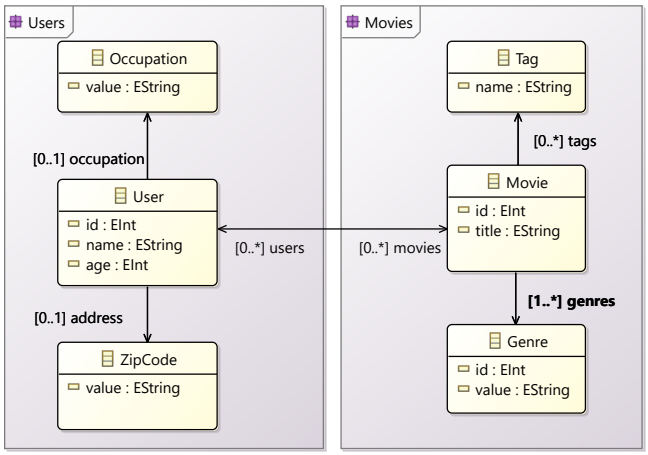

Figure 2 Excerpts of the Users and Movies metamodels

Figure 2 shows excerpts of the two initial metamodels for this example (expressed in EMF Ecore). `Users` are identified by an `id`, have a `name` and an `age`. Each `Movie` is identified by an `id`, has a `title`, and is also associated with a list of `Genres` and a list of `Tags`. The figure also shows the inter-model links between users and movies, which are only present in the historical dataset. A `User` may have watched several `Movies`, and a `Movie` may have been watched by several `Users`.

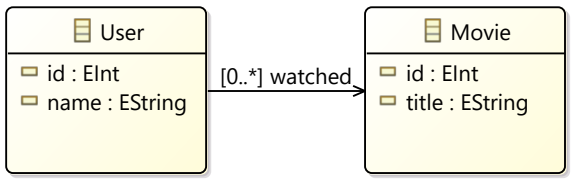

Figure 3 UsersAndMovies viewpoint metamodel

Figure 3 shows the desired viewpoint metamodel. We want to obtain a view that includes `ids` and `names` of `Users`, `ids` and `titles` of `Movies`, and a relation `watched` that lists for each user the movies she/he (probably) watched.

```
create view usersMovies as

select Users.User[id,name],
       Movies.Movie.*,
       Users.User join Movies.Movie
       as watched

from 'http://paper/movies' as Movies,
     'http://paper/users' as Users

where "t.tags
       ->exists(tag | tag.name = s.occupation.value)"
       for watched
```

Figure 4 VPDL file for defining a viewpoint for the running case using an OCL matching rule

Before our extension, the viewpoint definition language in EMF Views (VPDL) allowed only to express logical formulas for estimating inter-model links, called matching rules. Figure 4 shows a possible definition in standard VPDL for such view, with a simple matching rule.

The `select` part in VPDL is used to define which concepts and properties from which metamodel(s) will appear in the view (`*` means all properties). It also introduces new inter-model relations, i. e. the `watched` relation between `User` and `Movie` elements in our case. The `from` part allows users to declare the concerned metamodels, i. e.`Movies` and `Users` in our running example. Finally, the `where` part contains OCL-like expressions specifying matching rules for new inter-model relations, i. e. for `watched` in our case. In the example, we write a trivial rule that checks that among the `tags` of the movie (indicated as `t`, i. e. target of the possible link) there exists one whose `name` is equal to the `value` of the `occupation` for the user (indicated as `s`, i. e. source of the possible link). These OCL-like expressions are then automatically converted by EMF Views into an Epsilon Comparison Language (ECL)[2] matching rule that is actually used for computing the model view. It is worth mentioning that the letters "s" and "t" come from the syntax employed in EMF Views (inspired by the standard practice in ATL), where "s" denotes the Source and "t" denotes the Target of a relation. These variables are not defined directly in VPDL. Instead, they are defined in the generated ECL file.

Essentially, VPDL is a DSL for model view definition that includes a model query part. Having usability in mind, the design of this DSL was directly inspired by the well-known SQL constructs for defining views in databases (Bruneliere et al. 2015, 2020). Thus, while queries can actually be used in the ”WHERE” clause of VPDL for establishing new relations through the matching rules, the DSL also offers dedicated ”SELECT” and ”FROM” constructs.

As we said, this matching rule is not really representative of a real-world estimation. Moreover, the example already highlights important problems:

- A more realistic matching rule would require using a statistical programming library that is not available in VPDL.
- Engineers would have to use external tools to assess the validity of the rule against real-world data.

[2] https://www.eclipse.org/epsilon/doc/ecl/

In the following, we extend VPDL to define an automatic learning process for such matching rules from a set of previous examples, effectively bypassing these problems.

3. Proposed Approach

3.1. Overview

In the previous EMF Views approach (cf. the lower part of Figure 5), the view engineer has to provide two artifacts: a *Viewpoint definition* at the metamodel level and a *View definition* at the model level. In EMF Views, these two artifacts can be partially generated from a specification in VPDL, as described in the next subsection. Then, the *View Builder* takes these two artifacts as inputs and builds a virtual model that materializes the specified *Model View*. In our extended approach (cf. the upper part of Figure 5), we complement EMF Views with a new *View Learning* component to support the *View Builder* base component. A set of assignments for *GNN properties* is computed from the Viewpoint definition. It describes the architecture of the GNN and the hyperparameters for link prediction, including training and embedding. A *ML Specialist* can possibly edit the value of these properties, e. g. to fine-tune the learning step. *Training models* are also required, including existing links used as examples for learning. Such existing models can come from different sources, e. g. they can be collected from legacy projects. Then, the *View Learning* component takes these two artifacts as inputs and generates a trained HGNN. The set of inter-model links are computed by the *View Builder* component using the trained HGNN, before constructing the corresponding view.

Note that the EMF Views already supports delegating the computation of inter-model links to external tools. Hence, our proposed approach was able to reuse the standard structure of the *Viewpoint definition* and the standard *View Builder* component from EMF Views with no modifications. Moreover, the approach aims at decoupling the contributions of the *View engineer* and the *ML specialist*. Thus, the *ML specialist* can support the engineer by working on improving the accuracy and relevance of the inferred links without affecting the original *Viewpoint definition* and *View definition* made by the *View engineer*. Overall, we intend to make the use of ML as transparent as possible from the *View engineer* perspective. This way, she/he can focus solely on dealing with the modeling aspects while delegating ML integration and execution to our approach (and possible ML-specific optimizations to the ML specialist).

3.2. Extended ViewPoint Definition Language

We rely on the standard EMF Views for partially generating the *Viewpoint definition* and *View definition* from a specification in VPDL. Then, the View definition is manually completed to point to the actual resources, i. e., the contributing models. Additionally, our approach exploits our VPDL extension for generating default GNN architectures and hyperparameters (based on previous experiments) for the learning process.

Figure 6 shows a snippet of our viewpoint specification in Extended VPDL for our running example. In this new version, the *create* and *from* parts remain unchanged. However, the

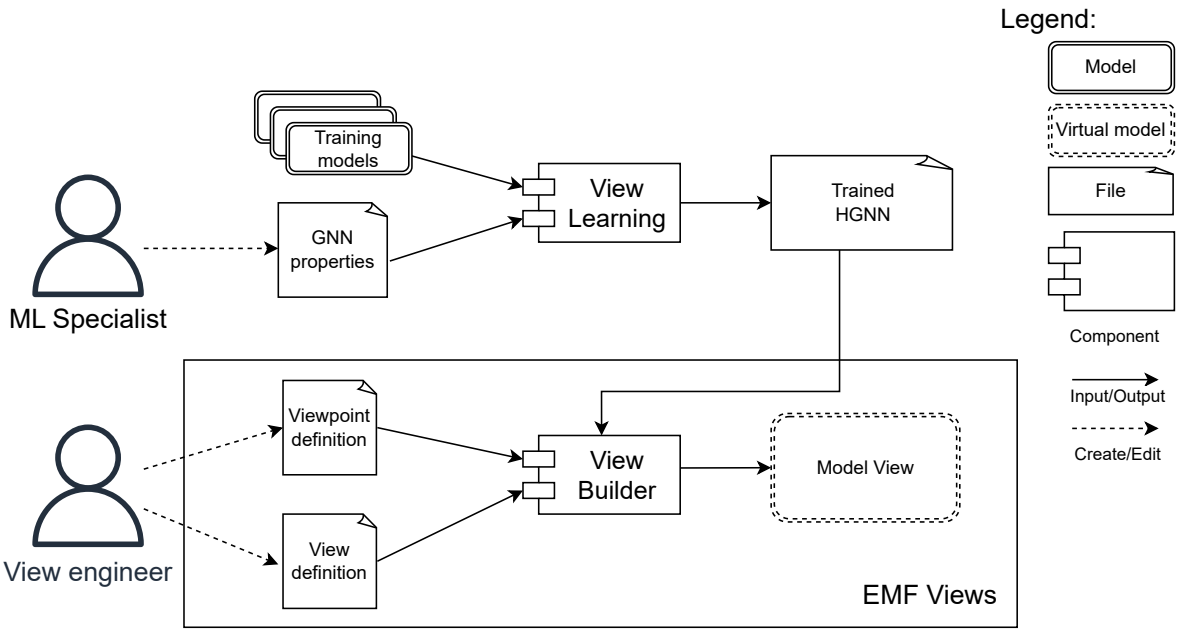

Figure 5 Overview of the proposed approach

```
create view usersMovies as

select Users.User[id, name],
       Movies.Movie.*,
       Users.User join Movies.Movie
       as watched

from 'http://paper/movies' as Movies,
     'http://paper/users' as Users

where  "{s.id}<~s.movies~>{t.id, t.genres.value}"
       for watched
```

Figure 6 Extended VPDL for the running case using ML

where part no longer contains an OCL-like expression but a specific expression indicating, for each inter-model relation, the properties of the two models and the training relation to be considered for learning. It contains:

- A set of navigation paths starting from the source of the relation `s`, indicating the properties that should be considered for characterizing the source element. In our case, `{s.id}` indicates that the learning system will only use the `id` of the user (and not the name, age, etc.).
- A set of navigation paths starting from the target of the relation `t`, indicating the properties that should be considered for characterizing the target element. In our case, `{t.id, t.genres.value}` indicates that the learning system will use the `id` of the movie and the list of its `genres`. Note that the navigation expression can navigate the model to access attributes of other model elements, e. g.`Genre`.
- A navigation path indicating an existing relation used as the source of examples. This path is always represented between the two previous sets, with a specific arrow notation. In our case, `<~s.movies~>` indicates that the learning system will consider the `movies` relation as the set of examples to learn from (in the direction starting from `s`).

3.3. View Learning Component

Figure 7 shows how the new *View Learning* component is organized internally. This component realizes the bridge between EMF models and ML heterogeneous graphs. We create one heterogeneous graph per relation to learn. This graph is a bipartite graph that contains only connections between nodes from the source and target models. The bipartite graph that corresponds to a given relation is constructed in the following way:

1. Embedding vectors (i. e., numerical representation in a lower dimension) for nodes are built by retrieving only the attributes involved in learning that relation (as indicated in VPDL), and by pre-processing them according to the GNN Properties.

2. An edge between two nodes is added if those nodes are connected by that relation in the training models.

In a second step, the component reads the *GNN properties* to instantiate a separate HGNN for each relation and performs training and inference. As shown in Listing 1, the *GNN properties* are serialized in a JSON file split into information blocks containing all necessary parameters for the HGNN definition, embedding, and training. We opted for the JSON format to allow for a straightforward modification of these parameters by the *ML specialist*.

The core elements of the HGNN, detailing its message-passing mechanisms and aggregation operations, are encapsulated by the `ARCHITECTURE` label. Each key within this block

[3] A LINKS_PATH property can be added when inter-model links are serialized in separate files.

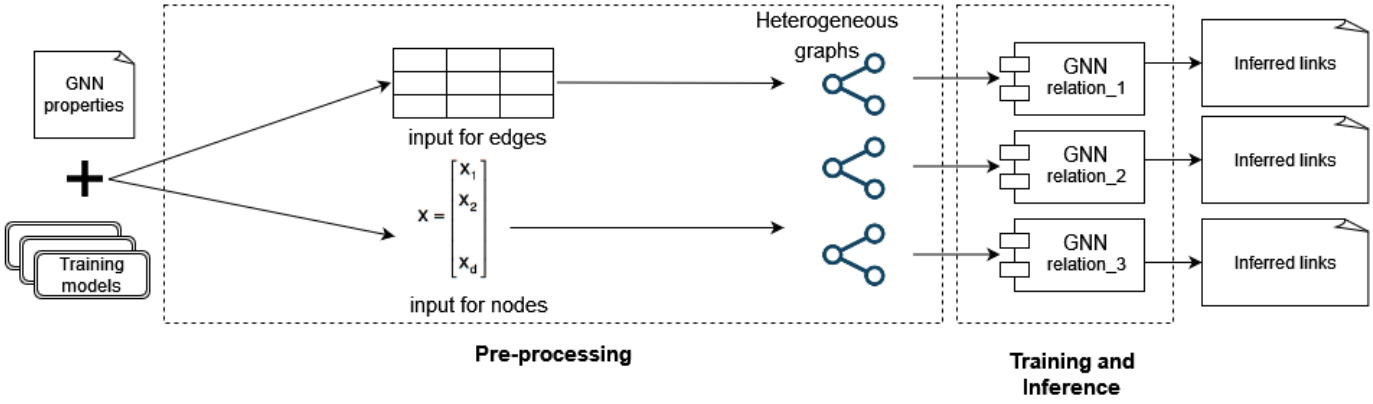

Figure 7 Structure of View Learning and inference

```
{
 "watched": {
  "ARCHITECTURE": {
    "OPERATOR": "SAGEConv",
    "CONVOLUTIONS": 2,
    "ACTIVATION": "relu",
    "HIDDEN_CHANNELS": 64,
    "CLASSIFIER": "dot_product"
  },
  "TRAINING_PARAMETERS": {
    "EPOCHS": 2,
    "LEARNING_RATE": 0.001,
    "ADD_NEGATIVE_TRAINING": false,
    "NEG_SAMPLING_RATIO": 2.0,
    "TRAINING_SPLIT" : 0.1,
    "VALIDATION_SPLIT" : 0.1,
    "SOURCE_MODEL_PATH": "users.xmi",
    "TARGET_MODEL_PATH": "movies.xmi" [3]
  },
  "EMBEDDINGS": {
    "s.id": "id",
    "t.id": "id",
    "t.genres.value": "enum"
  }
 }
}
```

Listing 1 GNN properties JSON file

corresponds to a single aspect of the HGNN's configuration: The `OPERATOR` key denotes the type of layer used for aggregation, while `CONVOLUTIONS` indicates the layer count. The `ACTIVATION` key specifies the activation function employed between layers, and the key `HIDDEN_CHANNELS` represents a numerical value determining the feature dimensions within hidden layers. Furthermore, the `CLASSIFIER` key indicates the function utilized to compute the final likelihood score for graph edges (i. e., edge decoder).

The usual hyper-parameters for standard neural network tasks as encapsulated by the `TRAINING_PARAMETERS` block, including epochs (`EPOCHS` key), learning rate (`LEARNING_RATE` key), and specific parameters for the link prediction task (e. g. addition of negative edges during training and strategies for edge splitting in training-test-validation). Additionally, this block encompasses the file paths for the *View Learning* component, namely the serialized models utilized during training. In case *training links* are stored in an independent file, this can be given too in an additional property (`LINKS_PATH`). Since we consider the inter-model link identification as a link prediction problem, it makes sense to split the graph into links (i. e., the training/-validation sets are split based on the links). Being so, the keys `ADD_NEGATIVE_TRAINING` and `NEG_SAMPLING_RATIO` are related to different strategies during this split, specifically regarding the inclusion or not of negative edges and the ratio for its inclusion. `TRAINING_SPLIT` and `VALIDATION_SPLIT`, as the name suggests, define how the links are split into training and validation sets, respectively. The user gives the paths to the models used for training through the keys `SOURCE_MODEL_PATH` and `TARGET_MODEL_PATH`.

The `EMBEDDINGS` block lists the properties specified for that relation in VPDL. For each one of them, we select their corresponding encoding scheme. For encoding, we currently support the following:

- `id` - Encoded as a lookup table for the element/node;
- `enum` - Encoded as a set of a fixed list. The strategy used is one-hot encoding;
- `string` - Uses a pre-trained language model to represent strings numerically. The user can indicate which model to use from the SentenceTransformers (Reimers & Gurevych

2019) library[4]. The current implementation is limited to the use of SentenceTransformers library. However, the use of other libraries, including MDE-specialized language models (Hernández López et al. 2023), are also possible with few adaptations;
– `number` - The value is simply cast to a float representation.

The definition of optimal default parameters is a problem that depends on various factors: the task you are working on, the characteristics of your graph data, or the GNN model's architecture being used, etc. Our work does not explicitly discuss the definition of criteria for these default parameters. However, our approach is designed precisely to simplify the work of the modeler and let these parameterization tasks to the ML specialist. Indeed, the ML specialist is more likely to have the necessary domain knowledge in order to make informed decisions.

It is also worth mentioning that some parameters, such as dataset split (e. g., 80 % for training and 20 % for validation) and the initial learning rate (often set to 0.001), are standardized within the ML community.

4. Implementation

This section describes the implementation of a prototype supporting the proposed approach. This prototype is open-source and publicly available[5]. We notably used the Eclipse Modeling Framework, as a basis of EMF Views and its *View Builder* component, and Xtext[6] for the implementation of VPDL. The *View Learning* component, mapping EMF models in a Python context and allowing different HGNN architectures, requires the use of the PyEcore[7] and PyTorch Geometric[8] Python libraries, respectively.

Figure 8 shows the essential files and their organization in the implemented prototype, applied to the running case. *Users.ecore* (the source of the link) and *Movies.ecore* (the target of the link) are the two metamodels considered in our viewpoint. *Users.xmi* and *Movies.xmi* are the two models that conform to the two previously mentioned metamodels (respectively) and that contain the actual data to build the target view. In practice, the view engineer writes in *my_view.vpdl* the VPDL specification of her viewpoint. The *viewpoint.eviewpoint* descriptor file is automatically generated from the VPDL specification. It contains pointers to the contributing metamodels and the EMF Views internal weaving model at the viewpoint-level (Bruneliere et al. 2020). The *my_view.eview* descriptor file is also automatically generated. It contains pointers to the corresponding viewpoint descriptor file, to the contributing models, and to the EMF Views internal weaving model at the view-level. In parallel, the ML specialist checks and adapts accordingly the *gnn_properties.json* complementary JSON file provided by default. Once done, she/he also provides the *user_movies.csv* inter-model relation file that is used by the *view_learning.py* Python code to build and train a proper *HGNN.model*. In addition, this Python code also produces a performance evaluation chart (c. f. section 5). Finally, the *relation_inference.py* Python code uses this trained ML model to generate the *watched.xmi* inter-model links file. This newly generated file can be ultimately used in the target view. To summarize, the *View Learning* component in our extended approach is implemented in these two *view_learning.py* and *relation_inference.py* Python files.

4.1. Limitations

On the VPDL side, we support the inference of several inter-model relations, each one learned by accessing an arbitrary set of properties of the training dataset.

For GNN properties, we currently support only a specific set of values in the `ARCHITECTURE` section:

– The `OPERATOR` is either "SAGEConv" or "HANConv";
– The number of `CONVOLUTION` layers is fixed to 2;
– The `ACTIVATION` function is either "relu" or "tanh";
– The `CLASSIFIER` is systematically "dot-product".

The current prototype is not fully integrated into the Eclipse/EMF environment. At this stage, the engineers must manually execute the provided Python scripts to launch training and inference. Additionally, the CSV file containing relations from previous models is considered as existing information adapted from legacy projects. All these elements will be progressively improved in the next versions of our prototype.

When dealing with the use of ML techniques in the MDE domain, an already known challenge is the lack of suitable datasets for training and benchmarking (Barriga, Rutle, & Heldal 2022). Our current evaluation is limited to some examples that focus on the use of domain models. Thus, it could be interesting to check the tool performance with other types of models having a more significant semantic gap.

5. Evaluation

To evaluate our approach, we consider three main aspects: 1) the efficiency of the HGNN for link prediction on real-world models, 2) how our approach compares to traditional solutions in terms of the number of lines of code (LOC) necessary to obtain a same result, and 3) the ability to learn pre-determined attribute-based matching rules. For 1) and 2), we apply our solution to the running example depicted in Section 2.3. For 3), we consider simple views on randomly generated models conforming to minimal metamodels and containing different relations.

5.1. Evaluation on the Running Example: Prediction Accuracy

To evaluate that our solution works efficiently enough for the running example, we created instances of the Users and the Movies metamodels (see Figure 2), using data from the *MovieLens* dataset provided by the GroupLens research lab (Harper & Konstan 2016). The first column of Table 1 presents the figures from the *ml-latest-small* dataset[9], including the time spent for

[4] https://www.sbert.net/docs/pretrained_models.html
[5] https://github.com/NaoMod/Support-ML-Relations-Model-Views
[6] https://www.eclipse.org/Xtext/
[7] https://github.com/pyecore
[8] https://pytorch-geometric.readthedocs.io/
[9] https://grouplens.org/datasets/movielens/latest/

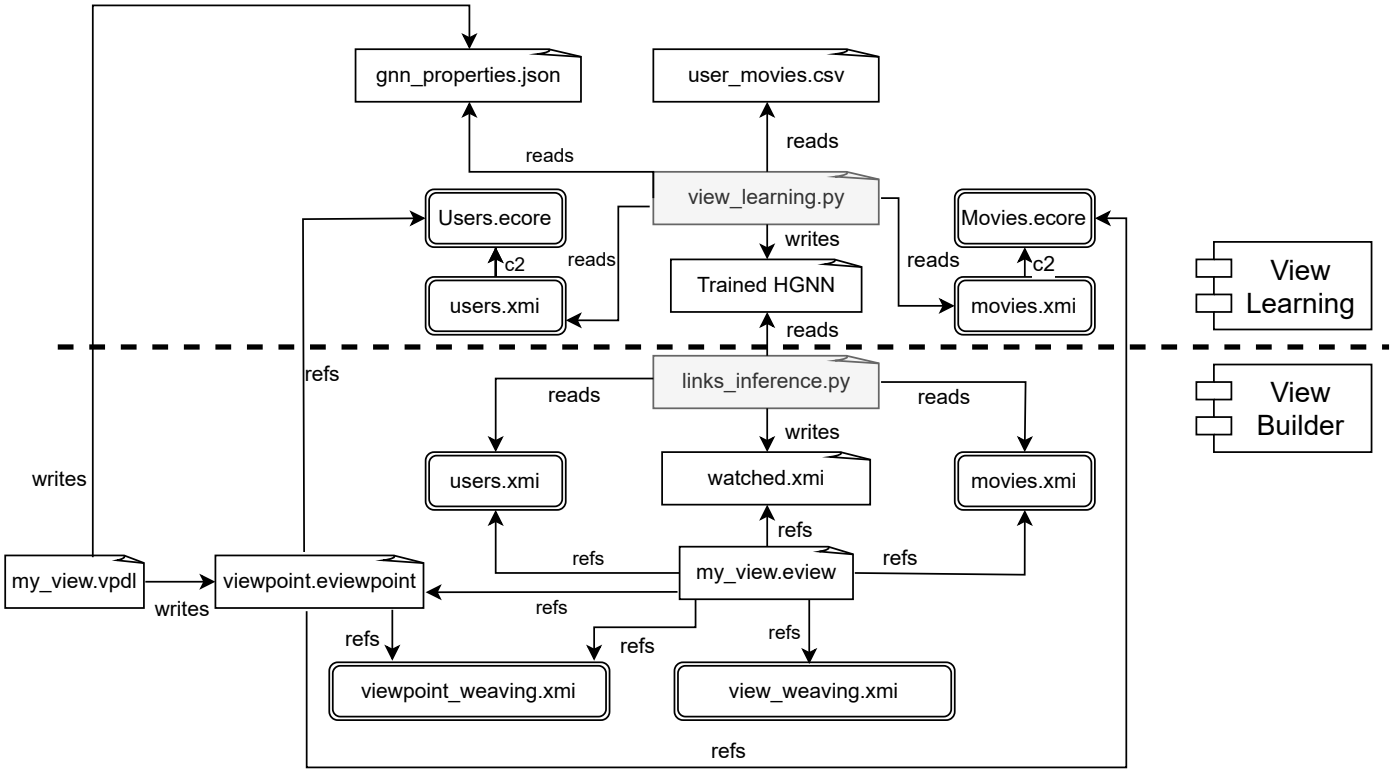

Figure 8 Main files in the prototype, applied to the running example (c2=conforms-to; refs=references)

training the HGNN.

	MovieLens	AB
nodes	610 Users, 9742 Movies	1000 As, 300 Bs
edges	100 863	5000
training time	58 s	∼50 s

Table 1 Dataset and training figures for MovieLens and AB

The evaluation metrics selected for the problem are the Receiver Operating Characteristic (ROC) curve and the respective area under the curve (AUC_ROC). The AUC_ROC measures the ability of a GNN model to distinguish between different classes by plotting the true positive rate against the false positive rate (Fawcett 2006). A value close to one denotes a good link prediction accuracy.

Figure 9 shows the resulting ROC curve for the `watched` relation described in the Section 4, for different values of the threshold. We can observe an $AUC\_ROC >= 0.9$, denoting good accuracy for this example.

5.2. Evaluation on the Running Example: LOC

With our approach, view engineers can now specify our example model view via concise statements in the extended VPDL, plus possibly modifying a JSON file for fine-tuning the ML configuration. Indeed, for our running case this only involves 39 LOC, consisting of 9 lines of VPDL code and 30 lines of a partially generated JSON configuration file.

We asked a proficient Python/PyTorch programmer to write an equivalent program with the same input files. This resulted in 385[10] lines of Python code (excluding blank lines and comments). As we stated, a main objective of our approach is to hide the use of ML code as much as possible so that engineers can focus on their modeling activities.

5.3. Learning Different Matching Rules

To evaluate this metric, we consider a different example from our running one. Figure 10 shows the two metamodels (in Ecore), each one containing one metaclass. The metamodel named *Left* has a class `A` with one numerical attribute `a` and one string attribute `s` besides its identifier. The metamodel named *Right* has only a class `B` with three numerical attributes `b`, `c` and `d`, and one string attribute `s`. Finally, a relation called `allBs` relates class `A` to class `B`. This example is named the *AB example*.

In this experiment, we evaluate the capacity of our tool to learn given matching rules on the *Left* and *Right* metamodels. We start defining three matching rules:

- $A.a = B.b$, i. e. two elements are related if they have the same numerical attribute
- $A.a = B.c * B.d$, i. e. two elements are related based on a simple mathematical operator (integer multiplication)
- $A.s.contains(B.s)$, i. e. two elements are related based on a simple string operator (containment).

[10] Available in the project repository

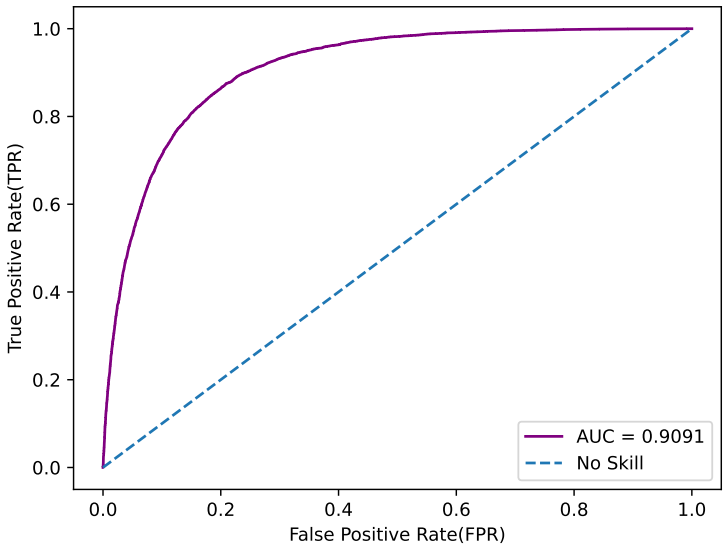

Figure 9 ROC curve for the running example

Threshold	FPR	TPR
5.8282	0.0000	0.0000
2.1575	0.0052	0.1364
1.7510	0.0105	0.2286
1.4524	0.0191	0.3095
1.1147	0.0314	0.4124
0.8142	0.0478	0.5066
0.4643	0.0742	0.6121
0.1082	0.1074	0.7065
-0.3633	0.1661	0.8081
-1.0493	0.2666	0.9044
-3.8462	0.7047	0.9945
-6.2632	0.9176	0.9994

Table 2 ROC curve for the running example (tabular form)

Note that these matching rules do not consider the structural aspects of the model. We evaluated the capability of the HGNN to learn from attribute values per node type. Given one of the three matching rules, the sample models are generated with random values, but the generator guarantees that all instances of `A` have at least a matching element of type `B` for that matching rule. Finally, the generator connects a random ratio of matching elements, leaving many missing links, i. e. connections that should exist but were not created. Once the data set is created, we follow the same approach explained in the previous sections.

Figure 11 shows one ROC curve per matching rule. These curves show good results ($0.85 <= AUC\_ROC <= 0.94$) considering the small amount of data we used (5000 connections for 4000 nodes). They show that, by using standard hyper-parameters for HGNN definition and training, our tool can learn simple matching rules on attribute values.

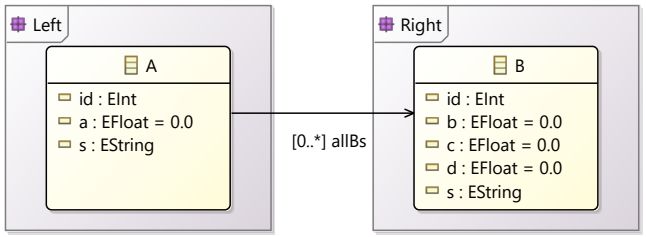

Figure 10 Metamodels for the AB example

6. Related work

6.1. Model Views

Preserving the consistency and synchronization of a given model view implies having efficient ways of creating and handling the relations between the various models contributing to this view. For example, the EMF Views solution (Bruneliere et al. 2015) relies on a combination of model virtualization and matching rules to dynamically initiate model views and such inter-model links. Guerra et al. propose a formal approach for specifying the relations between modeling languages through the use of a pattern-based language for inter-modeling with a focus on model-to-model transformations, model matching, and model traceability (Guerra et al. 2010). Although their approach is not explicit about model views, their definition of inter-model links can be considered as a potential mechanism to compute views based on patterns. Quite differently, the ModelJoin solution (Burger et al. 2016) proposes to rely on a metamodel generator and higher-order transformations to compute these links on the fly. In the same vein, Boronat (Boronat 2019) intro-

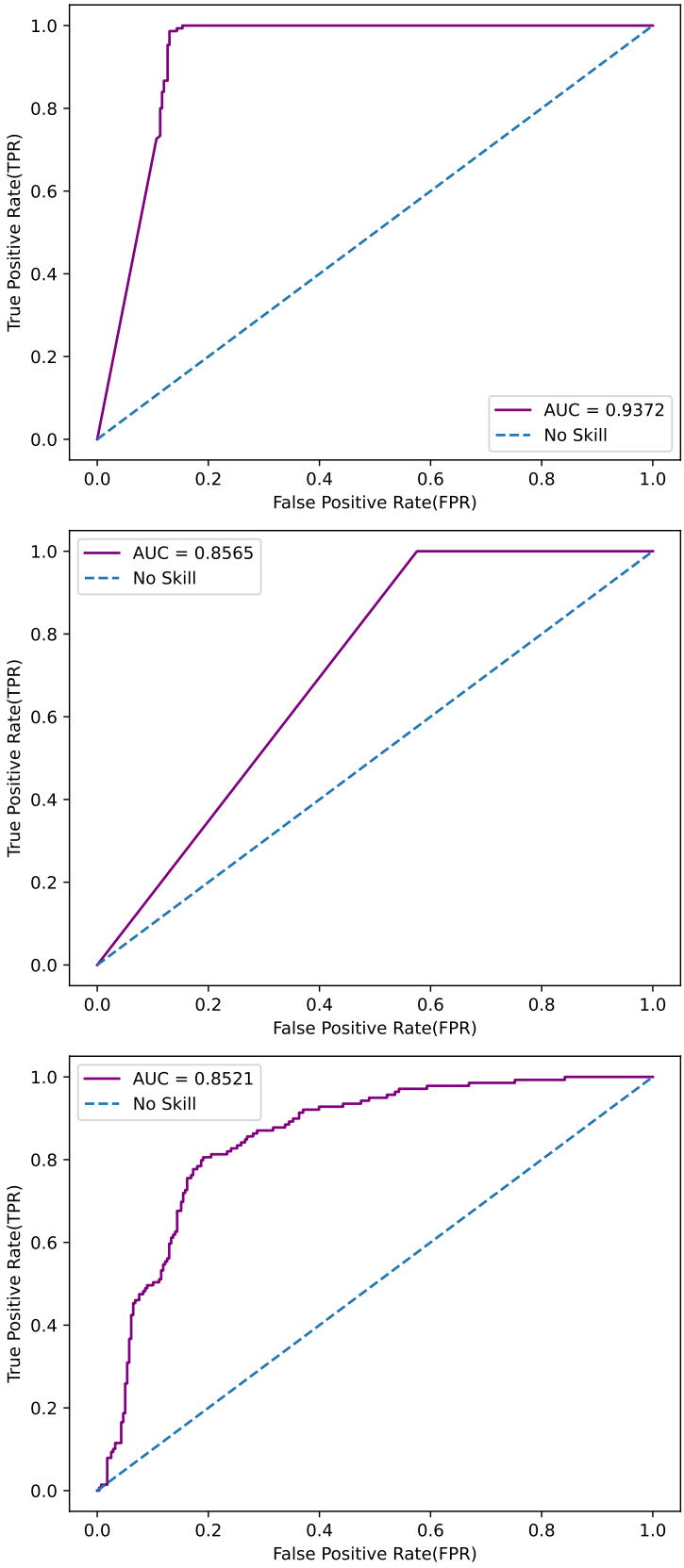

Figure 11 ROC curves for different AB relations: $A.a = B.b$ at the top, $A.a = B.c * B.d$ in the middle, and $A.s.contains(B.s)$ at the bottom

duces an independent model view synchronization mechanism based on transformation techniques to deal with the interaction between the original models and the view. To the best of our knowledge, AI techniques have not yet been directly used in such a model view context. We intend to make a step in this direction with the work presented in this paper.

6.2. ML for MDE

ML has been widely used to support different Software Engineering activities (Shafiq et al. 2021). It has also been applied in the area of modeling approaches and languages (Burgueño, Cabot, Wimmer, & Zschaler 2022) where AI/ML has already been identified as a relevant way of addressing several important challenges (Bucchiarone et al. 2020). Model consistency management is a fundamental issue in model-based engineering and thus creates the need for relevant model repair techniques (Macedo et al. 2017). As they require a *smart* automation, such techniques are natural candidates for AI applications (Barriga, Rutle, & Heldal 2022). For example, different approaches propose the use of rule-based ML (Nassar et al. 2017), decision trees (Kretschmer et al. 2018), or Reinforcement Learning (RL) (Iovino et al. 2020; Barriga, Heldal, et al. 2022), in order to find the best sequence of actions for repairing a given model and reaching a sufficient quality level. Groner et al. propose the use of different ML techniques which are not based on Neural Networks to predict the execution time of ATL model transformations (Groner et al. 2023). However, the use of these AI techniques does not appear to be the most appropriate in the context of our present work. Indeed, we do not intend to fix inconsistent models or to improve execution performance. We rather want to better federate consistent models by automatically inferring new links between them. On the (meta)modeling side, Weyssow et al. (Weyssow et al. 2022) propose assistance for metamodeling activities based on pre-trained language models. Our proposal follows a similar research line.

6.3. Learning Constraints and Transformations

Among the work that applies ML to MDE, research on learning constraints and transformations is the most similar to our effort. Dang & Cabot (Dang & Cabot 2015) describes the InferOCL tool dedicated to the automatic inference of OCL constraints for conceptual models and metamodels based on examples. Their objective is to increase the precision of domain description. While we are also learning OCL expressions, we are more interested in particular expressions that express matching between elements from two separate models. On the model transformation side, there is significant literature on Model Transformations by Example (MTBE) (Kappel et al. 2012; Strommer 2008), a user-friendly approach that aims at generating transformation rules from example of inter-model mappings. More recently, Burgueño et al. proposed a generic neural network architecture to support the automated inference of model-to-model and model-to-text transformations (Burgueño, Cabot, Li, & Gérard 2022). Their objective is notably to limit potential implementation errors. Since views can be implemented by transformations, we pay particular attention to these approaches. Compared to these research efforts, we are addressing a more limited problem, computing the probability of the existence of a certain link. This allows us to use HGNNs, a solution that showed high performance in this specific problem.

6.4. GNNs for MDE

López & Cuadrado (López & Cuadrado 2023) focus on achieving structural realism by utilizing a deep auto-regressive model combining a GNN and a Recurrent Neural Network (RNN). Compared with existing generators, their evaluation demonstrates its superior structural realism, consistency, diversity, and scalability in generating new models. In parallel, Di Rocco et al. (Di Rocco et al. 2021) develop and experiment with the MORGAN tool, a GNN-based recommender system designed to assist modelers in specifying metamodels and models. Although different, these kinds of AI applications (especially the last one) are closer to the context of our present work. The related work shows that GNNs appear to be particularly adapted to such scenarios where additional data must be inferred from existing models, notably related to their structural aspects.

7. Conclusions and Future Work

In this paper, we proposed an approach for automatically inferring inter-model links in the context of model views. This approach relies on ML techniques, particularly on HGNNs. Its objective is to support view engineers in specifying viewpoints and corresponding views when inter-model links can be inferred from a set of existing examples. The proposed approach intends to lower the barrier of using ML in the context of model views. It also facilitates collaboration with ML specialists who can help the view engineer to improve the accuracy of the link prediction.

To this end, we refined and extended the existing VPDL model view specification language from EMF Views to properly integrate the automated generation and use of view-dedicated HGNNs. In practice, we implemented our approach by combining the Eclipse-based EMF Views solution with two Python libraries, PyEcore and PyTorch Geometric, dedicated to model handling and HGNNs, respectively. Based on this implementation, we conducted a first set of experiments that showed promising results concerning the automated inference of inter-model links, also reducing the number of ML code to be written by engineers.

However, there is still room for future work regarding various aspects of our solution. Among the different tracks to be explored, we can notably mention the following:

- *Modeling language*: Our Extended VPDL and its companion files is a first version we plan to continue working on. An immediate next step is to develop a dedicated language for the previously presented JSON file for GNN Properties and to integrate it more smoothly with VPDL. To this purpose, we will study the potential reuse of existing work related to DSLs for supporting ML activities (Giner-Miguelez et al. 2023; Rajaei et al. 2021).
- *Systematically evaluating the learning capability*: We showcased the expressiveness of our language and approach by implementing a few model views with different inter-model relations. However, additional experiments

are still required to systematically assess which matching rules could be effectively replaced by a trained HGNN. Moreover, we also plan to experiment with our approach more generally in the context of the inference of matching rules within model transformations.

- *Support more variability in GNN architectures*: While the current GNN properties allow us to effectively address the running example, more options to describe GNNs and training processes could help with edge cases. For example, while we support the aggregation function selection, we do not support standard internal parameters for each function, e. g. learning additive bias, application of linear transformation after activation layers, normalization of output, etc.. These values can vary among different aggregation functions.
- *Improving the inference capability of the approach*: HGNN performance could be further optimized through offline hyper-parameter tuning or by exploring alternative architectures. We may allow users to choose among threshold selection strategies (e. g., based on the provided AUC_ROC metric or adding new complementary metrics (Koyejo et al. 2014)). In some cases, these improvements will require further extending VPDL. For instance, we want to support the specification of meta-paths by the user in order to guide the discovery of complex relations.

On the evaluation aspect, our approach can benefit by incorporating a comprehensive analysis of each of the main HGNN default parameters, including possible integration with other ML automation techniques (i. e. AutoML) for hyperparameter optimization. Another improvement for future work includes evaluating different examples from the MDE domain, including when these models have a higher semantic gap between them, distinct from our running example. These new datasets will potentially enable us to evaluate the relation between the established metrics and the models' size. In a different direction, future work can also include using different ML techniques to complement other aspects of the definition and use of Model Views, enabling comparisons between the use of GNNs and other potential approaches.

Acknowledgments

This work was partially funded by the AIDOaRt European project, an ECSEL Joint Undertaking (JU) project under grant agreement No. 101007350, and by a complementary grant from the French region Pays de la Loire.

References

Afzal, W., Bruneliere, H., Di Ruscio, D., Sadovykh, A., Mazzini, S., Cariou, E., … Smrz, P. (2018, September). The MegaM@Rt2 ECSEL project: MegaModelling at Runtime – Scalable model-based framework for continuous development and runtime validation of complex systems. *Microprocessors and Microsystems*, *61*, 86–95. Retrieved 2023-06-30, from https://www.sciencedirect.com/science/article/pii/S014193311830022X doi: 10.1016/j.micpro.2018.05.010

Barriga, A., Heldal, R., Rutle, A., & Iovino, L. (2022, October). PARMOREL: a framework for customizable model repair. *Software and Systems Modeling*, *21*(5), 1739–1762. Retrieved 2022-11-09, from https://doi.org/10.1007/s10270-022-01005-0 doi: 10.1007/s10270-022-01005-0

Barriga, A., Rutle, A., & Heldal, R. (2022, February). AI-powered model repair: an experience report—lessons learned, challenges, and opportunities. *Software and Systems Modeling*. Retrieved 2022-03-15, from https://link.springer.com/10.1007/s10270-022-00983-5 doi: 10.1007/s10270-022-00983-5

Boronat, A. (2019, November). Code-First Model-Driven Engineering: On the Agile Adoption of MDE Tooling. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 874–886). (ISSN: 2643-1572) doi: 10.1109/ASE.2019.00086

Bruneliere, H., Burger, E., Cabot, J., & Wimmer, M. (2019, June). A feature-based survey of model view approaches. *Software & Systems Modeling*, *18*(3), 1931–1952. Retrieved 2022-01-26, from http://link.springer.com/10.1007/s10270-017-0622-9 doi: 10.1007/s10270-017-0622-9

Bruneliere, H., de Kerchove, F. M., Daniel, G., Madani, S., Kolovos, D., & Cabot, J. (2020, July). Scalable model views over heterogeneous modeling technologies and resources. *Software and Systems Modeling*, *19*(4), 827–851. Retrieved 2022-01-11, from http://link.springer.com/10.1007/s10270-020-00794-6 doi: 10.1007/s10270-020-00794-6

Bruneliere, H., Muttillo, V., Eramo, R., Berardinelli, L., Gómez, A., Bagnato, A., … Cicchetti, A. (2022, October). AIDOaRt: AI-augmented Automation for DevOps, a model-based framework for continuous development in Cyber–Physical Systems. *Microprocessors and Microsystems*, *94*, 104672. Retrieved 2023-06-30, from https://www.sciencedirect.com/science/article/pii/S0141933122002022 doi: 10.1016/j.micpro.2022.104672

Bruneliere, H., Perez, J. G., Wimmer, M., & Cabot, J. (2015, October). EMF Views: A View Mechanism for Integrating Heterogeneous Models.. Retrieved 2022-01-07, from https://hal.inria.fr/hal-01159205 doi: 10.1007/978-3-319-25264-3_23

Bucchiarone, A., Cabot, J., Paige, R. F., & Pierantonio, A. (2020, January). Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, *19*(1), 5–13. Retrieved 2022-07-19, from https://doi.org/10.1007/s10270-019-00773-6 doi: 10.1007/s10270-019-00773-6

Burger, E., Henss, J., Küster, M., Kruse, S., & Happe, L. (2016, May). View-based model-driven software development with ModelJoin. *Software & Systems Modeling*, *15*(2), 473–496. Retrieved 2023-07-07, from https://doi.org/10.1007/s10270-014-0413-5 doi: 10.1007/s10270-014-0413-5

Burgueño, L., Cabot, J., Li, S., & Gérard, S. (2022, February). A generic LSTM neural network architecture to infer heterogeneous model transformations. *Software and Systems Modeling*, *21*(1), 139–156. Retrieved 2023-07-07, from https://doi.org/10.1007/s10270-021-00893-y doi: 10.1007/s10270-021-00893-y

Burgueño, L., Cabot, J., Wimmer, M., & Zschaler, S. (2022, June). Guest editorial to the theme section on AI-enhanced model-driven engineering. *Software and Systems Modeling*, *21*(3), 963–965. Retrieved 2023-07-07, from https://doi.org/10.1007/s10270-022-00988-0 doi: 10.1007/s10270-022-00988-0

Dang, D.-H., & Cabot, J. (2015). On Automating Inference of OCL Constraints from Counterexamples and Examples. In V.-H. Nguyen, A.-C. Le, & V.-N. Huynh (Eds.), *Knowledge and Systems Engineering* (pp. 219–231). Cham: Springer International Publishing. doi: 10.1007/978-3-319-11680-8_18

Dehghani, M., Kolahdouz-Rahimi, S., Tisi, M., & Tamzalit, D. (2022, June). Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. *Software and Systems Modeling*, *21*(3), 1115–1133. Retrieved 2022-11-09, from https://doi.org/10.1007/s10270-022-00977-3 doi: 10.1007/s10270-022-00977-3

Di Rocco, J., Di Sipio, C., Di Ruscio, D., & Nguyen, P. T. (2021, October). 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). doi: 10.1109/MODELS50736.2021.00016

Fawcett, T. (2006, June). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. Retrieved 2023-06-28, from https://www.sciencedirect.com/science/article/pii/S016786550500303X doi: 10.1016/j.patrec.2005.10.010

Giner-Miguelez, J., Gómez, A., & Cabot, J. (2023, August). A domain-specific language for describing machine learning datasets. *Journal of Computer Languages*, *76*, 101209. Retrieved 2023-07-07, from https://www.sciencedirect.com/science/article/pii/S2590118423000199 doi: 10.1016/j.cola.2023.101209

Groner, R., Bellmann, P., Höppner, S., Thiam, P., Schwenker, F., & Tichy, M. (2023, April). Predicting the performance of atl model transformations. In *Proceedings of the 2023 acm/spec international conference on performance engineering* (p. 77–89). New York, NY, USA: Association for Computing Machinery. Retrieved from https://dl.acm.org/doi/10.1145/3578244.3583727 doi: 10.1145/3578244.3583727

Guerra, E., de Lara, J., Kolovos, D. S., & Paige, R. F. (2010). Inter-modelling: From theory to practice. In D. C. Petriu, N. Rouquette, & y. Haugen (Eds.), *Model driven engineering languages and systems* (p. 376–391). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-16145-2_26

Harper, F. M., & Konstan, J. A. (2016, January). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*, *5*(4), 1–19. Retrieved 2023-06-15, from https://dl.acm.org/doi/10.1145/2827872 doi: 10.1145/2827872

Hernández López, J. A., Durá, C., & Cuadrado, J. S. (2023, October). Word embeddings for model-driven engineering. In *2023 acm/ieee 26th international conference on model driven engineering languages and systems (models)* (p. 151–161). Retrieved from https://ieeexplore.ieee.org/document/10344175?denied= doi: 10.1109/MODELS58315.2023.00036

Iovino, L., Barriga Rodriguez, A., Rutle, A., & Heldal, R. (2020). Model Repair with Quality-Based Reinforcement Learning. *19*. Retrieved 2022-07-19, from https://hvlopen.brage.unit.no/hvlopen-xmlui/handle/11250/2737208 (Accepted: 2021-04-12T07:41:21Z Publisher: AITO — Association Internationale pour les Technologies Objets) doi: 10.5381/JOT.2020.19.2.A17

Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., & Wimmer, M. (2012). Model Transformation By-Example: A Survey of the First Wave. In A. Düsterhöft, M. Klettke, & K.-D. Schewe (Eds.), *Conceptual Modelling and Its Theoretical Foundations: Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday* (pp. 197–215). Berlin, Heidelberg: Springer. Retrieved 2023-07-07, from https://doi.org/10.1007/978-3-642-28279-9_15 doi: 10.1007/978-3-642-28279-9_15

Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=SJU4ayYgl

Koyejo, O. O., Natarajan, N., Ravikumar, P. K., & Dhillon, I. S. (2014). Consistent Binary Classification with Generalized Performance Metrics. In *Advances in Neural Information Processing Systems* (Vol. 27). Curran Associates, Inc. Retrieved 2023-07-07, from https://proceedings.neurips.cc/paper_files/paper/2014/hash/30c8e1ca872524fbf7ea5c519ca397ee-Abstract.html

Kretschmer, R., Khelladi, D. E., & Egyed, A. (2018, May). An automated and instant discovery of concrete repairs for model inconsistencies. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (pp. 298–299). New York, NY, USA: Association for Computing Machinery. Retrieved 2022-07-19, from https://doi.org/10.1145/3183440.3194979 doi: 10.1145/3183440.3194979

Liben-Nowell, D., & Kleinberg, J. (2003, November). The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management* (pp. 556–559). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-06-23, from https://dl.acm.org/doi/10.1145/956863.956972 doi: 10.1145/956863.956972

López, J. A. H., & Cuadrado, J. S. (2023, April). Generating Structurally Realistic Models With Deep Autoregressive Networks. *IEEE Transactions on Software Engineering*, *49*(4), 2661–2676. (Conference Name: IEEE Transactions on Software Engineering) doi: 10.1109/TSE.2022.3228630

Lü, L., Medo, M., Yeung, C. H., Zhang, Y.-C., Zhang, Z.-K., & Zhou, T. (2012, October). Recommender systems. *Physics Reports*, *519*(1), 1–49. Retrieved 2023-06-23, from https://www.sciencedirect.com/science/article/pii/S0370157312000828 doi: 10.1016/j.physrep.2012.02.006

Macedo, N., Jorge, T., & Cunha, A. (2017, July). A Feature-Based Classification of Model Repair Approaches. *IEEE Transactions on Software Engineering*, *43*(7), 615–640. (Conference Name: IEEE Transactions on Software Engineering)

doi: 10.1109/TSE.2016.2620145
Nassar, N., Radke, H., & Arendt, T. (2017). Rule-Based Repair of EMF Models: An Automated Interactive Approach. In E. Guerra & M. van den Brand (Eds.), *Theory and Practice of Model Transformation* (pp. 171–181). Cham: Springer International Publishing. doi: 10.1007/978-3-319-61473-1_12
Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). DeepWalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701–710). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-06-23, from https://dl.acm.org/doi/10.1145/2623330.2623732 doi: 10.1145/2623330.2623732
Pontes Miranda, J., Bruneliere, H., Tisi, M., & Sunyé, G. (2024, April). Towards the Integration Support for Machine Learning of Inter-Model Relations in Model Views. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24)*. Avila, Spain. doi: 10.1145/3605098.363614
Rajaei, Z., Kolahdouz-Rahimi, S., Tisi, M., & Jouault, F. (2021, June). A DSL for Encoding Models for Graph-Learning Processes.. Retrieved 2023-07-07, from https://hal.science/hal-03252919
Reimers, N., & Gurevych, I. (2019, November). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 3982–3992). Hong Kong, China: Association for Computational Linguistics. Retrieved 2023-06-29, from https://aclanthology.org/D19-1410 doi: 10.18653/v1/D19-1410
Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009, January). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80. (Conference Name: IEEE Transactions on Neural Networks) doi: 10.1109/TNN.2008.2005605
Shafiq, S., Mashkoor, A., Mayr-Dorn, C., & Egyed, A. (2021). A Literature Review of Using Machine Learning in Software Development Life Cycle Stages. *IEEE Access*, *9*, 140896–140920. (Conference Name: IEEE Access) doi: 10.1109/ACCESS.2021.3119746
Shi, C., Wang, X., & Yu, P. S. (2022a). Introduction. In C. Shi, X. Wang, & P. S. Yu (Eds.), *Heterogeneous Graph Representation Learning and Applications* (pp. 1–8). Singapore: Springer. Retrieved 2023-06-22, from https://doi.org/10.1007/978-981-16-6166-2_1 doi: 10.1007/978-981-16-6166-2_1
Shi, C., Wang, X., & Yu, P. S. (2022b). Platforms and Practice of Heterogeneous Graph Representation Learning. In C. Shi, X. Wang, & P. S. Yu (Eds.), *Heterogeneous Graph Representation Learning and Applications* (pp. 285–310). Singapore: Springer. Retrieved 2023-06-27, from https://doi.org/10.1007/978-981-16-6166-2_10 doi: 10.1007/978-981-16-6166-2_10
Strommer, M. (2008). *Model Transformation By-Example* (Doctoral dissertation, Vienna University of Technology, Vienna). Retrieved 2023-07-07, from https://publik.tuwien.ac.at/files/PubDat_166165.pdf
Tang, X., Wang, Z., Qi, J., & Li, Z. (2019). Improving code generation from descriptive text by combining deep learning and syntax rules. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, *2019-July*, 385–390. (ISBN: 1891706489) doi: 10.18293/SEKE2019-170
Weyssow, M., Sahraoui, H., & Syriani, E. (2022, June). Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling*, *21*(3), 1071–1089. Retrieved 2023-02-07, from https://doi.org/10.1007/s10270-022-00975-5 doi: 10.1007/s10270-022-00975-5

About the authors

James Pontes Miranda is a PhD student in the NaoMod Team at IMT Atlantique, LS2N (CNRS) in Nantes, France. His current research focus is on the use and adaptation of Machine Learning techniques in the context of model views and their concrete applications within the AIDOaRt project. You can contact the author at james.pontes-miranda@imt-atlantique.fr.

Hugo Bruneliere is a senior researcher in the NaoMod Team at IMT Atlantique, LS2N (CNRS) in Nantes, France. His research interest is in designing and adapting model-based techniques and architectures to address software and systems engineering issues. His works have been recently applied in the context of complex Cyber Physical Systems or in the area of Cloud/Fog Computing (among others). He has also a long-term experience on leading research activities and working in the context of large collaborative projects with both industrial and academic partners (including the AIDOaRt project). You can contact the author at hugo.bruneliere@imt-atlantique.fr.

Massimo Tisi Massimo Tisi is an associate professor in the Department of Computer Science of the Institut Mines-Telecom Atlantique (IMT Atlantique, Nantes, France), and deputy leader of the NaoMod team, LS2N (UMR CNRS 6004). His research interests revolve around software and system modeling, domain specific languages, and applied logic. He contributes to the design of the ATL model-transformation language and investigates the application of deductive verification techniques to model-driven engineering. You can contact the author at massimo.tisi@imt-atlantique.fr.

Gerson Sunyé is an associate professor at the University of Nantes (France) in the domain of software engineering and distributed architectures and the scientific leader of the NaoMod team. He received the PhD degree in Computer Science from the University of Paris 6 in 1999 and the Habilitation from Nantes University in 2015. He has 4 years of industry experience in software development. He is the author of several papers in international conferences and journals in software engineering. His research interests include software testing, design patterns and large-scale distributed systems. You can contact the author at gerson.sunye@ls2n.fr.