Encoding Conceptual Models for Machine Learning: A Systematic Review

Syed Juned Ali, Aleksandar Gavric, Henderik A. Proper, Dominik Bork

To appear in:

Companion Proceedings of the 26th International Conference on Model Driven Engineering Languages and Systems, (MODELS-C 2023). 5th Workshop on Artificial Intelligence and Model-driven Engineering

Final version available soon:

www.model-engineering.info

Encoding Conceptual Models for Machine Learning: A Systematic Review

Syed Juned Ali
Business Informatics Group, TU Wien
syed.juned.ali@tuwien.ac.at

Aleksandar Gavric
Business Informatics Group, TU Wien
aleksandar.gavric@tuwien.ac.at

Henderik Proper
Business Informatics Group, TU Wien
henderik.proper@tuwien.ac.at

Dominik Bork
Business Informatics Group, TU Wien
dominik.bork@tuwien.ac.at

***Abstract*—Conceptual models are essential in Software and Information Systems Engineering to meet many purposes since they explicitly represent the subject domains. Machine Learning (ML) approaches have recently been used in conceptual modeling to realize, among others, intelligent modeling assistance, model transformation, and metamodel classification. These works encode models in various ways, making the encoded models suitable for applying ML algorithms. The encodings capture the models' structure and/or semantics, making this information available to the ML model during training. Therefore, the choice of the encoding for any ML-driven task is crucial for the ML model to *learn* the relevant contextual information. In this paper, we report findings from a systematic literature review which yields insights into the current research in machine learning for conceptual modeling (ML4CM). The review focuses on the various encodings used in existing ML4CM solutions and provides insights into i) *which* are the information sources, ii) *how* is the conceptual model's structure and/or semantics encoded, iii) *why* is the model encoded, i.e., for which conceptual modeling task and, iv) *which* ML algorithms are applied. The results aim to structure the state of the art in encoding conceptual models for ML.**

***Index Terms*—Machine learning, Model-driven engineering, Model Encoding, Systematic Literature Review**

I. Introduction

Conceptual modeling (CM) explicitly captures (descriptive and/or prescriptive) domain knowledge where a domain, in an enterprise and information systems engineering context, is anything that is being modeled, including—but not limited to—business processes, information structures, business transactions, and value exchanges, enabling domain understanding and communication among stakeholders [1]. Model-driven engineering (MDE) is a software development approach that emphasizes the use of models[1] as the primary artifacts throughout the entire software development lifecycle. These models can be automatically transformed and refined to generate executable code, documentation, and other artifacts [2].

Applying Machine Learning (ML) techniques i.e., Deep Learning (DL) and Natural Language Processing (NLP), on data provided by conceptual models has gained much attention in supporting various conceptual modeling tasks such as intelligent modeling assistants [3], model completion [4], model transformation [5], metamodel repository management, and model domain classification [6], [7]. Furthermore, there is a potential to apply ML to publicly available sources of high quality (F.A.I.R. principles [8]) models to enable reuse, adaptation, and (collaborative) learning, as well as empirical modeling research.

ML on conceptual models aims to "learn" generalized patterns that capture the explicit mapping between the conceptual model's elements and the domain concepts represented by them. In other words, the trained ML model should be able to answer *what* the conceptual model represents in terms of the "meaning" of the domain concepts and model elements. ML-based solutions for conceptual modeling follow a specific pattern of first encoding the conceptual model's semantics in a representation suitable for training ML models. Then, the ML models are trained to learn the knowledge encoded in conceptual models to support CM tasks like metamodel element prediction and domain classification. ML models typically aim to learn generalized patterns from an input dataset by *utilizing* a certain *encoding* of the knowledge represented by the conceptual models. Therefore, the encoding constraints what can an ML model learn from the available knowledge in the model. The contextual information that captures representative semantics of the data needs to be accessible to the ML model during training for the ML model to learn semantically rich patterns. Current ML-based CM solutions primarily rely on the lexical terms (i.e., *names*) used as labels on modeling language primitives (e.g., classes, relations, attributes) to capture the models' contextual semantics. This leads to a situation where the natural language (NL) semantics of the primitives are encoded. However, additional sources of semantics such as *structural semantics*, the *metamodel semantics*, and the CM elements' *ontological semantics* are left implicit.

Therefore, various issues arise depending upon the requirements that need to be addressed before applying ML to CM tasks. Firstly, the *source*s of relevant information need

[1]Throughout the paper, we will use the term 'model' to relate to a conceptual model and 'ML model' to relate to machine learning models.

to be decided, i.e., *which* information sources need to be made available to the ML model to learn. The source of information could be structural, i.e., graph-based properties of the conceptual model and/or semantic, e.g., lexical terms, metamodel, and ontological semantics. Secondly, *how* should the model structure and semantics be encoded to be used by the ML model during training? Finally, based on the selected information for a task and the selected encoding, *which* ML model should be used to train on the encoded models? This topic still needs to be well understood and has not been explored in depth. Therefore, we conducted a Systematic Literature Review (SLR) to comprehensively analyze how the issues mentioned above are dealt with by the state-of-the-art and find some crucial insights that would allow us to draw some associations between the different encodings on the one hand and different purposes, modeling languages, and ML models used on the other. Finally, we make our complete results available[2], including the links to the model datasets used.

The remainder of this paper is structured as follows: Section II presents the related works. Section III describes our SLR research methodology, including the research questions we address. In Section IV we present the responses to the research questions. We discuss our overall findings in Section V before we concluded this paper with Section VI.

II. Related Work

In recent years there has been a surge in the works of combining AI with conceptual modeling. Based on a study [9], machine learning has been the area of artificial intelligence most applied with conceptual modeling. These works focus on using AI to do conceptual model processing, i.e., automated processing of the information present in a conceptual model to assist the modeler in modeling tasks.

Lopez et al. [7] present a comparative study of different ML classification techniques that automatically label models stored in model repositories. They compare different ML models (e.g., Feed-Forward Neural Networks, Graph Neural Networks, and K-Nearest Neighbors) with varying model encodings (TF-IDF, word embeddings, graphs, and paths). However, several differences to our work need to be pointed out. Firstly, they do not discuss the *source* of information, i.e., *which* information is made accessible to the model, and do not differentiate *how* the structure and semantics of the model are encoded. In our work, we separate the source of structural and semantic sources of information and subdivide the semantic sources further into linguistic, metamodel, and ontology-based sources. Secondly, their study focuses on model domain classification applications, which do not comprehensively understand the relationship between the encoding and the applications requiring model encoding. E.g., they report that even though structural encoding schemes based on graphs should be superior based on the rationale that they are a good match for the graph-based nature of software models, simpler encodings that do not require graph-based encoding perform better. However, it is not surprising that domain classification would not require model structure information because the lexical terms of the model sufficiently capture the information required for the domain classification task. Therefore, the choice of encoding is task-dependent and encodings should be selected based on the task details.

The research area of ML4CM and model-driven engineering (MDE) is still recent. Therefore, there is a lack of related work. Many papers pragmatically use ad-hoc model encodings specific to their application requirements, often lacking a systematic and comprehensible elaboration on the encoding choice and its alternatives. For e.g., Clariso et al. [10] present graph kernels as a generalized model encoding for clustering software modeling artifacts and improve the efficiency and usability of various software modeling activities, e.g., design space exploration, testing, verification, and validation but without a systematic review of other encodings. It is important to note that our study is not a comparative study of encodings. Instead, with our SLR, we aim to provide a better understanding of the literature in relation to model encoding such that researchers and practitioners interested in applying ML to conceptual models can make an informed decision for encoding their models depending on the task they need to solve.

III. Research Method

Our Systematic Literature Review (SLR) followed the research method introduced by Kitchenham and Charters [11]. The SLR aims to analyze the state-of-the-art in the context of model encodings in ML4CM. In the remainder of this section, we will describe the steps involved in our SLR.

A. Defining the research scope

The paper at hand aims to respond to the following main research questions: **RQ1**: *Which* information present in the conceptual model is used for ML training?; **RQ2**: *How* is the information encoded for ML training?; **RQ3**: *How* does the ML purpose correlate with the used *encoding*?; and **RQ4**: *How* does the ML model correlate with the *encoding*?

In responding to RQ1, we will investigate which information provided by a conceptual model (e.g., structural, semantic) is incorporated in current ML4CM approaches and which sources of relevant data are used during ML training. For responding to RQ2, we zoom-in on the different encodings available to represent the model information in a format suitable for ML. RQ3 is responded to by separately investigating the correlation between the purpose i.e., the CM task to be solved, and the encoding and modeling language used. RQ4 is responded to by determining how the choice of ML models relates to the purpose and the chosen encoding.

B. Conduct Search

We conducted a larger and much more generic and inclusive Systematic Mapping Study (SMS) about *Conceptual Modeling* and *Artificial Intelligence* (results are reported in [9]) using

[2]https://goo.by/HEA7D

the following logically structured search query in eq. (1). Instead of starting from scratch and developing a separate query, we filtered out the documents relevant to our study from the SMS. We understand that this approach can be seen as a limitation. However, we chose this alternative for two reasons: i) our query in the SMS is very inclusive (see eq. (1)), and ii) we had already done a detailed review and were able to easily exclude papers which had the contribution in the direction of AI towards CM (AI4CM). Note that due to the nature of our query, we do not include the works that do not apply ML in their approach. This implies that the works that e.g., propose non-ML-based similarity metrics based on the structural features of the model graph or the semantics of the model elements are not within the scope of this work. We aim to conduct a broader review to cover such cases in the future.

$$Q = (\vee CM_i) \wedge (\vee AI_j), \text{where} \quad (1)$$

$CM_i \in \{$*"conceptual modeling"*, *"metamodel"*, *"meta-model"* *"domain specific language"*, *"modeling formalism"*, *"modeling tool"*, *"modeling language"*, *"modeling method"*, *"model driven"*, *"model-driven"*, *"mde"*$\}$

$AI_j \in \{$*"artificial intelligence"*, *"ai"*, *"machine learning"*, *"ml"*, *"deep learning"*, *"dl"*, *"neural network"*, *"genetic algorithm"*, *"smart"*, *"intelligent"*$\}$

C. Screening papers

After executing the query on January 16, 2023, we followed the steps shown in Fig. 1 to screen the papers relevant to our study. The top portion of Fig. 1 shows the screening steps of the SMS. We got 647 relevant papers with mappings. In the current work, we focus on the literature that encodes models for applying ML methods. We applied four exclusion criteria for further filtering: exclude papers that **EC-1:** contribute from CM to AI; **EC-2:** focus the conceptual model creation using text-to-model, image-to-model transformation approaches because we need the model as the starting artifact for model encoding; therefore, this excludes papers that, e.g., apply NLP to generate domain-specific models from text; **EC-3:** involve "genetic algorithms" because genetic algorithms do not learn from the models' data unlike ML approaches; and **EC-4:** did not report any model encodings-related information. Furthermore, we ensured we did not miss important papers by doing a forward/backward search. After the screening, we had **37** remaining relevant papers.

D. Search for keywords in abstracts

After filtering the 37 papers, we carefully reviewed all the papers and applied a classification over several attributes of our RQs. Table I shows all the attribute dimensions for classifying papers and describes each attribute with its possible values. A conceptual model captures information in its structural and semantic data[3]. We consider that the structural and semantic data can be encoded explicitly or implicitly depending on the encoding. For example, N-grams encoding, specific

[3]We refer to semantics as non-structural data like lexical terms in the model

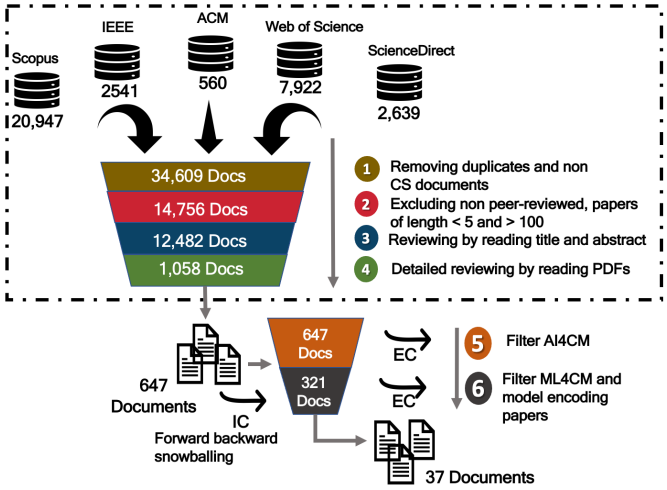


Fig. 1: SLR relevant papers screening

metrics such as the number of classes, and the number of cyclic dependencies in a model, can implicitly encode the model structure [12] without explicitly encoding the graph as a network of nodes and edges. Similarly, metrics related to the semantic data, e.g., *type* of model elements like the number of relations of type generalization, can implicitly capture the semantic information [13]. Therefore we classify the model structure and semantic data in the encoding as *explicit*, *implicit*, and *not encoded*. The source of structural data is the model's graph structure; however, semantic data comes from the lexical terms associated with the model's elements (entities, relationships, attributes) in natural language and from external sources. We restrict the external sources to a model's metamodel (e.g., EClassifier, EAttribute for ECore, Aspect, Layer for ArchiMate models) and external domain ontology (e.g., WordNet), and foundational ontology (e.g., UFO). Furthermore, in Table I, we show the classification of ML models based on different types of ML models. However, we focus on individual models as well in our data analysis (see Section IV-C). Based on our classification scheme, Fig. 2 shows the overall CM encoding process. We aim to get insights into this process using our SLR.

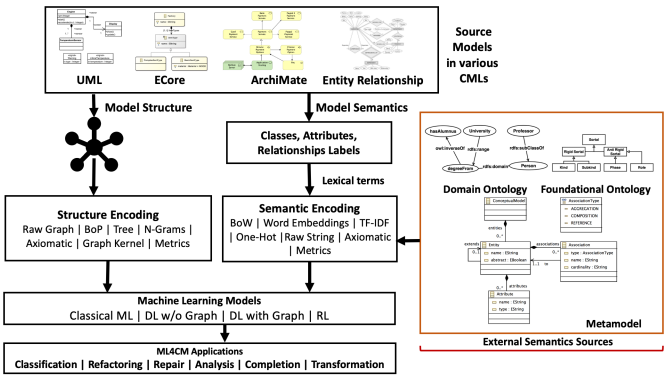


Fig. 2: Steps involved in ML4CM

TABLE I: Classification scheme keywords description

Attribute	Description	Values
Model Structure	If the model's graph structure is encoded or not.	Explicit, Implicit, Not Used
Structural Encoding	The model structure encoding type.	Raw Graph, Tree-based, Graph Kernel, Bag of Paths, Axiomatic, N-grams, Manual Metrics
Semantic Data	If the semantic data in the model is encoded.	Explicit, Implicit, Not Used
Metamodel Semantics	If the metamodel semantics are captured in the encoding.	Yes, No
Ontological Semantics	If the model terms are annotated with ontological semantics and further used in model encoding.	Yes, No
Semantic Encoding	The model semantics encoding type.	BoW Word Embeddings, BoW TF-IDF, Raw BoW, One-hot, Raw String, Manual Metrics
Modeling Purpose	The ML-based application for which the model is encoded.	Analysis, Classification, Completion, Refactoring, Repair, Transformation
ML Model	The ML model used in the paper to train on the encoded models' data.	Classical Machine Learning, Deep Learning without Graph, Deep Learning with Graph, Reinforcement Learning

IV. Findings

In the following, we present the results of our data analysis and in effect, respond to the RQs defined above.

A. Response to RQ1 – Which information present in the conceptual model is used for ML training?

In Fig. 3 we show which model information is used to encode models. The figure shows that using explicit information sources, i.e., the lexical terms of the model elements and the model structure are most commonly used to encode models. Several works also encode metamodel information (10 papers) and ontological semantics (5 papers) explicitly. We see that the natural language lexical terms of model elements have the highest contribution to the semantic data (26 papers), followed by metamodel-based semantics (11 papers, 10 explicit, and 1 implicit), and the least used are ontological semantics (6 papers, 5 explicit, and 1 implicit). Some works use metamodel-level information, e.g., element types like EPackage, and EClassifier, but in most cases, the ML model does not use the metamodel information. The model's labels are user-defined labels not rooted in any domain or foundational ontology. Rooting the model in an external ontology requires ontology alignment, which requires additional effort. Therefore, these aspects are consistent with the results in Fig. 3.

Only a few cases are implicitly encoded, i.e., using metrics and keywords. Four papers implicitly encode the model structure and only two implicitly capture the metamodel and ontological semantics. Encoding the graph structure directly allows ML models to jointly consider global and local structure [14] rather than using selected metrics focusing only on global graph structural information. Similarly, using lexical terms directly seems logical over using a manually curated set of metrics. ML models can capture the latent correlations between model elements' terms that a manually curated set of metrics might miss. Fig. 3 also shows that the literature focused on the model's semantics because the model's structure is not used in a model's encoding in almost half the cases (15 papers) whereas only five papers lack semantic data encoding.

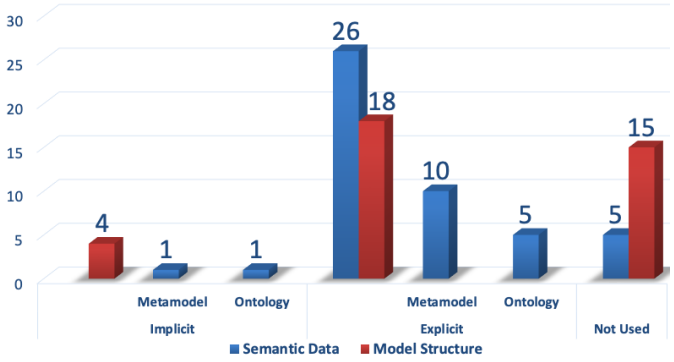


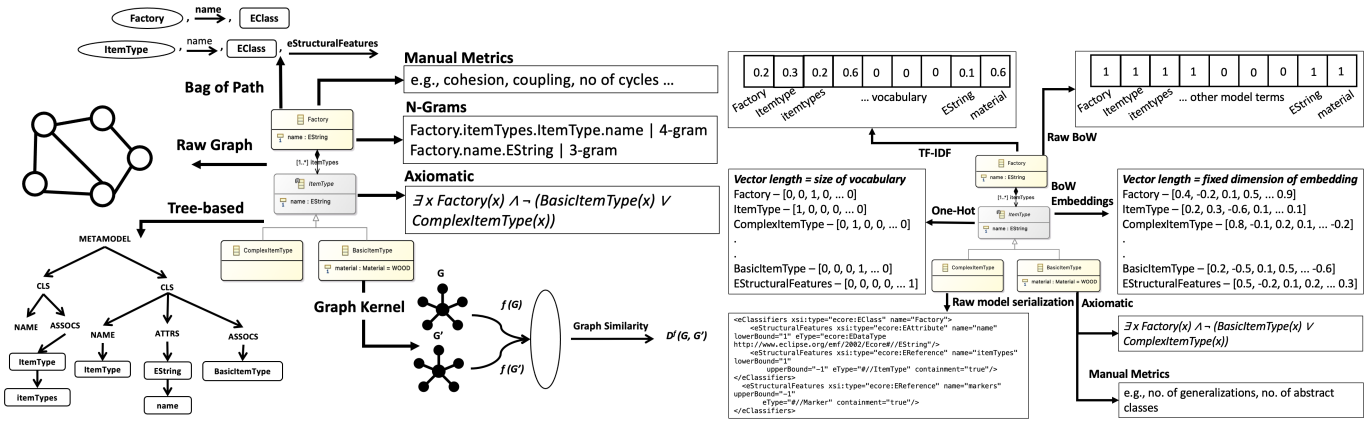
Fig. 3: Data source distribution for model encoding

B. Response to RQ2 – How is the model information encoded?

In the following, we zoom-in into *how* the model's structure and semantic data are encoded. After classifying all the papers with the corresponding structural and semantic encoding, we found seven different types of structural and semantic encodings as shown in Fig. 4.

1) Structural Encodings: – Graph structural, i.e., encodings that explicitly capture the model's structure using the graph structure information (cf. Fig. 4a) include: i) *Raw Graph*; ii) *Tree-based* encoding where each model is represented as an independent tree, the root contains the keyword MODEL, and its children are the model elements, which can be either OBJECTSs, ASSOCIATIONs [5]; and iii) *Bag of Path* (BoP) where the paths of fixed lengths capturing the nodes and edges between two nodes of the model graph are stored [15]. The implicit encodings include i) *manually selected metrics* depending on the user requirements; ii) *n-grams*, which captures the sequence of vertices' labels in the model of length n-1 [12]; iii) *Axiomatic* representation which represents the model in terms of a set of axioms [16]; and iv) *Graph kernels* to embed sub-structures of models into some features [17].

2) Semantic Encodings: – Semantic data encodings are visualised in Fig. 4b. The lexical terms of the models are encoded in the following ways: i) *Model serialization* which uses the model directly in its XML format; ii) *One-hot* encoding where each lexical term in the encoding is represented in the form of a fixed-sized vector with all zeros except a single one



(a) Structural Encodings (b) Semantic Encodings

Fig. 4: Visualization of different model encodings

corresponding to the lexical term, where the size of the vector is the total size of the vocabulary; iii) Raw *Bag-of-words* (BoW) where the model is represented as a vector containing all lexical model terms; iv) *Term Frequency-Inverse Document Frequency* (TF-IDF) vector where the term frequency along with inverse document frequency of the lexical terms is calculated and then the model is represented with the TF-IDF value of each lexical term in the model; v) *BoW embeddings* where each word is represented in the form of a fixed-sized vector of arbitrary length where the values in the vector for each word are produced by a language model pre-trained on a general or domain-specific data corpus; vi) *manual metrics* that use some specific keywords (e.g., keyword "set", "get" in the model serialization) metrics to implicitly capture the model semantics; and vii) *Axiomatic* representation, which is the same as in the case of structural encoding.

3) Encodings' usage analysis: – We show the analysis of the different encoding pairs used in both the structural and semantic dimensions in Table II. We note several key things from the table. Firstly, the "No Encodings" for the model structure column has the most papers. This is consistent with the fact that 15 out of 37 papers did not include structural encodings (see Fig. 3) and only used semantic data encoding, with TF-IDF as the most common encoding. BoW word embeddings and TF-IDF are vector-embeddings-based encodings and are the most common choice to embed the semantic data (19 out of 37). This choice seems logical because if one needs to capture the correlations between the lexical terms of the model, its metamodel, and any ontological semantics associated with the model, then techniques like TF-IDF and pre-trained language models (LM) can capture these correlations more effectively as these techniques learn (in case of LMs), compute (in case of TF-IDF) these word embeddings over a large vocabulary, thereby providing a more contextual encoding. Several works apply NLP normalization techniques like stop word removal, stemming, and lemmatization before encoding the lexical terms with TF-IDF or LMs. Interestingly, we found that some works do not even apply NLP techniques while using lexical terms. There are different reasons for this, such as i) using string comparison metrics like Levenshtein Distance[4]; ii) defining own metrics for calculating the similarity between labels of elements [27], [28] which does not require any normalization using NLP; iii) renaming all the tokens that are not keywords to a closed set of words (for instance, classes' names are A, B, C, ..., attribute names are x, y, z, ..., etc.) [5]; or iv) using only specific keywords as lexical terms where the NL semantics of the terms are not relevant [16], [27]. Word embeddings from large language models (LLMs) (GPT[5], BERT[6]) can encode lexical terms to get nuanced contextualized word embeddings. Therefore, several recent works use LLMs-based word embeddings.

Thirdly, the model encoded as a raw graph is the most common encoding technique for capturing the model's structural information. However, the number of works overall is significantly less (8 out of 37). Moreover, we see that raw graph as a structural encoding and BoW Embeddings as the semantic encoding is a frequent combination. Further analysis showed that cases that use this combination benefit from capturing both the structural and semantic information, e.g., learning a vector representation of a model [44], and characterizing a model generator [45]. Other path-based encodings, such as N-grams and Bag-of-Path (BoP), that can encode the model as a set of paths are not frequent. This seems to be due to these encodings' limitation in sufficiently capturing the model structure. Finally, several works have used manually selected metrics and axiomatic representation to encode the model's structure and semantics. In these cases, authors, instead of using the ML model, design their task-specific metrics without applying any encoding.

C. Response to RQ3 – How does the ML purpose correlate with the used encoding and modeling language?

In the following, we elaborate on the different purposes of model encodings for ML4CM.

[4]https://en.wikipedia.org/wiki/Levenshtein_distance
[5]https://openai.com/research/gpt-4
[6]https://huggingface.co/docs/transformers/model_doc/bert

TABLE II: Structural and Semantic Encodings across all relevant papers

Semantic Encoding	Structural Encodings								
	No Encodings	**Manual Metrics**	**Axiomatic**	**N-grams**	**BoP**	**Graph Kernel**	**Tree-based**	**Raw Graph**	**Total**
No Encoding	N/A	[18]	[19]	✗	[20]	[17]	✗	[21], [22]	**6**
Manual Metrics	✗	[13], [23], [24]	✗	✗	✗	✗	✗	✗	**3**
Axiomatic	[16]	✗	✗	✗	✗	✗	✗	✗	**1**
Serialized model	[25], [26]	✗	✗	✗	✗	✗	✗	✗	**2**
Raw BoW	[27], [28]	✗	✗	✗	✗	✗	✗	✗	**2**
One-hot	[29], [30]	[31]	✗	✗	✗	✗	✗	[32]	**4**
TF-IDF	[6], [33]–[37]	✗	✗	[12], [38]	[15]	✗	✗	[39]	**10**
BoW Embeddings	[40], [41]	✗	✗	✗	[42]	✗	[4], [5]	[39], [43]–[45]	**9**
Total	**15**	**5**	**1**	**2**	**3**	**1**	**2**	**8**	**37**

1) Extracted Purposes: After carefully reading the papers, we classified each paper in one of the following categories: i) *Analysis*—if ML is applied to do some model analysis e.g., discovering patterns in the model [20], ii) *Classification*—if ML is applied to classify the encoded model based on user-defined model similarity criteria into one of the user-defined classes e.g., domain classification of metamodels [7], [34], iii) *Completion*—if ML is used to autocomplete a partial model, recommend elements to the modeler, e.g., NLP-based model autocompletion [41], iv) *Refactoring*—if ML is used to support model refactoring, e.g., model-driven bug report visualisation [43], v) *Repair*—if ML is used to repair a partially broken model e.g., [16], and vi) *Transformation*—if ML is used for model transformation e.g., [5].

2) Purpose Analysis: Fig. 5 shows classification and completion as the most common ML-based applications to conceptual models. ML methods can efficiently find patterns in conceptual models which can be used to characterize and classify models. Therefore, classification has been applied for e.g., conceptual model search and automatic metamodel clustering. In the case of model completion, which involves predicting the next element given a partial model, BoW Embeddings is the most used encoding. Model encodings that capture the contextual information of model elements and further allow similarity comparison help predict the next element more accurately due to the available contextual information. Word embeddings capture information locally (i.e., within the model) and globally across a large dataset of models, even across domains, acting as a rich contextual data source. Therefore, it is consistent that most works used word embeddings to capture semantic data for model completion. Moreover, structural encoding (as a raw graph) is also mostly used for model completion. Predicting the next element in a model involves predicting the next node or edge in the graph. Therefore it would help the ML model to know the structural information during prediction. Furthermore, a model-driven project involves several consecutive transformations, and automating model transformation operations can reduce the time-to-market of project development and improve its quality [5]. This explains the next highest frequency of model transformation as the ML4CM purpose.

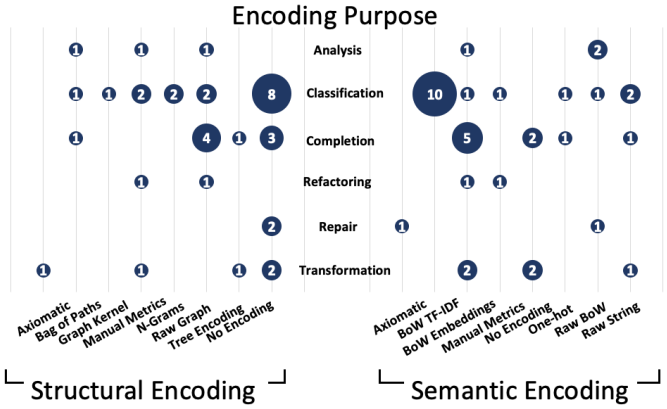


Fig. 5: Relationship of Purpose with model encoding

D. Response to RQ4 – How does the ML model correlate with the encoding and purpose?

In this research question, we focus on the used ML model and potential correlations with the model encoding.

1) ML model classification: We divided the ML models into four classes depending on the type of learning architecture as follows: i) *Classical Machine Learning* which do not involve any Deep Learning architecture, the ML models in this category include XGBoost[7], CatBoost[8], Support Vector Machine (SVM), Random Forest, Apriori association rules, K-nearest neighbors, Integer Linear Programming, and Naive Bayes; ii) *Deep Learning without Graph* which includes DL architectures that do not explicitly capture the graph structure of the data with the Transformer[9], Long-Short-Term-Memory (LSTM) and Feed Forward Neural Networks as ML models; iii) *Deep Learning with Graph* which includes DL architectures that capture the graph structure with Graph Neural Networks (GNN) and Graph-aware Attention Networks (GaAN); and iv) *Reinforcement Learning* (RL) which includes ML models like Markov Decision Process (MDP).

2) ML model usage analysis: Fig. 6 shows the relationship between the used ML models with the model encodings. Fig. 6 (left) shows the different ML categories and the used models

[7]https://xgboost.readthedocs.io last accessed 04.07.2023
[8]https://catboost.ai/ last accessed 04.07.2023
[9]https://huggingface.co/docs/transformers/ last accessed: 24.08.2023

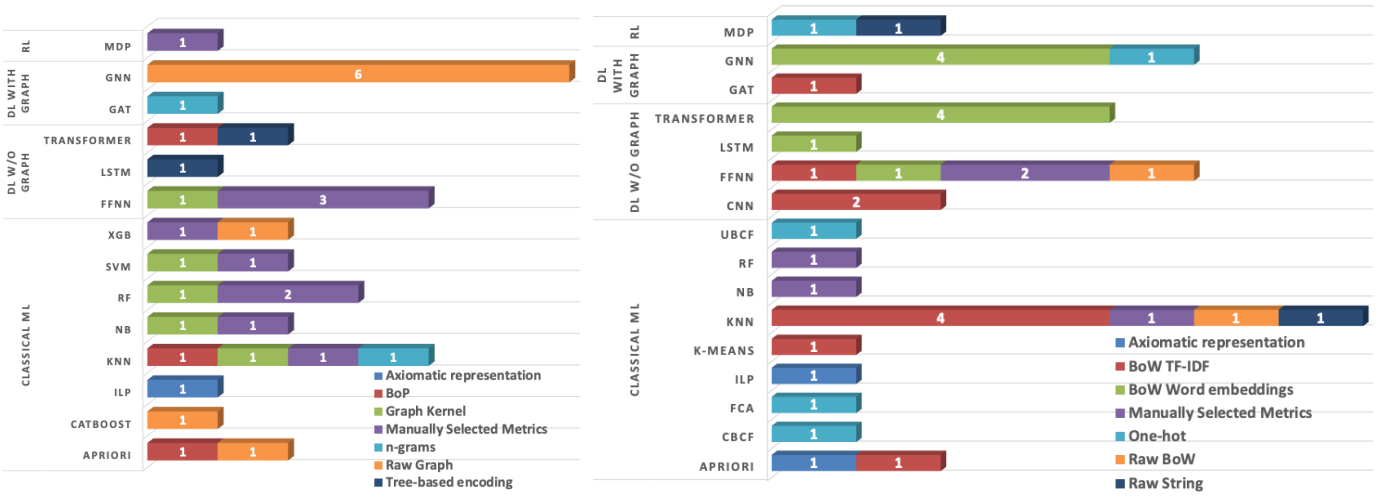


Fig. 6: Structural (left) and semantic (right) encodings with ML models

to encode the model structure. The plot shows that GNN and FFNN are the most used models to capture the model structure. Moreover, GNNs use a raw graph as model encoding, making GNNs suitable for learning the structural and semantic information from the conceptual model's graph encoding. Furthermore, tree-based encodings are used to serialize the model as a sequence of tokens [4], [5] to make the model encoding suitable for DL models like Transformers and LSTM (which do not explicitly capture the model structure) for model completion and transformation. However, tree-based encodings in the current works do not capture longer dependencies, i.e., exceeding an element's direct neighbors. In contrast, raw graph with GNNs allows capturing such information to larger depths, which explains the higher frequency of the combination of raw graphs with GNNs. Multiple models are used with Graph Kernel encoding, where Graph Kernels transform the model into a set of features and use them to apply graph similarity metrics with ML models like SVM, Naive Bayes, and Random Forest. BoP encoding stores the model as a collection of paths such that the paths (or part of models) thereby allow model similarity comparisons using different ML approaches like KNN, Apriori association rules, or even complex DL models like Transformers as shown in Fig. 6. Note that N-grams are quite similar to BoP in capturing the sequence of vertices that can capture relationships but do not capture complex relationships compared to BoP [46]. However, it is interesting to note that the N-grams encoding is also used with GaAN, where GaAN compensates for the limitations of N-grams of not capturing the complex relationships by capturing longer graph structural dependencies.

Fig. 6 shows that KNN and GNNs are frequently used for semantic encoding. KNN seems to be a common choice due to its simplicity of finding similarity measures of models where the nearest neighbor of a model is considered a similar model based on the model encoding. The similarity measure enables efficient model comparison and thereby, classification. Furthermore, KNN is most frequently used with BoW TF-IDF encoding. The common encodings used with GNNs are BoW word embeddings. This shows that GNNs can capture structural semantics by encoding the graph's structural aspects and semantic data using generalized semantically rich word embeddings. Moreover, FFNN and KNN are frequently used as general-purpose ML models to encode semantic data with different kinds of encodings. Transformers have been used only with BoW word embeddings because of the Transformer architecture's capability of fine-tuning generalized word embeddings for a given context. Therefore, the papers that use Transformers first use the generalized word embeddings from the pre-trained language models like BERT and then fine-tune their embeddings for their task [4]. Finally, user and content-based collaborative filtering (UBCF and CBCF) use a one-hot encoding for model elements recommendation [30] and a K-Means with BoW TF-IDF encoding for model classification [36]. There are other ML approaches such as Random Forest (RF), Naive Bayes (NB), and Inductive Linear Programming that are not usually used in ML4CM research.

V. Discussion

This section summarizes our findings, discusses insights, and reflects on the remaining research gaps we observed. We see in Fig. 3 that there is a lack of metamodel and ontological semantics contribution towards the "meaning" of model elements. We consider this lack a first research gap. The relationship of the model elements with the domain is not sufficiently captured by only the lexical terms represented as BoW, TF-IDF, or word embeddings. Using only the NL semantics of words leads to missing out on the contextual semantics provided by the model's metamodel capturing model elements' *types* and ontological semantics capturing domain concepts. Moreover, providing only *type* level information i.e., an element is a *Class* or *Relationship*, not the relationship between the *types* on the metamodel level also hides information. Without encoding metamodel or ontological semantics, the ML model misses out on learning *type* level semantics, the relationship between *types*, the properties of *types* (why is a class abstract, when does a class need to be abstract), common software design patterns, what kind of a foundational ontological stereotype should the class have—all of which

is important information that makes the conceptual model a semantically rich artifact.

We further see in Fig. 3 and Table II that in many cases structural encodings are not used. We consider this as a second research gap. Moreover, graph encoding techniques like Graph Kernel, which capture local and global neighborhood structures, are underrepresented and can be used to add more structural information to the encoding.

We acknowledge that in our SLR, we have not provided a performance analysis of each of the encodings related to different purposes. However, comparative performance evaluation is difficult because of the lack of standardized datasets for specific purposes and specific modeling languages. There are further no baselines to test the performance of different encodings systematically. In our analysis, we found that out of all the works that make their dataset public, all the datasets are different except for [47] which shows a lack of standardized datasets. Recently, Lopez et al. [7] performed a comparative analysis of ML encodings for the domain classification task. However, there is a need to do similar studies for other ML4CM tasks because, as we see from our analysis, the choice of encoding is task-dependent.

VI. Conclusion

In this paper, we provided an SLR-based detailed analysis of the various encodings used in the context of machine learning for conceptual modeling (ML4CM) i.e., using ML methods to support CM tasks. We zoomed into *what* information from the model is encoded, i.e., its semantics and/or structure. We then analyzed *how* the information is encoded, thereby identifying 14 different encodings for structural and semantic aspects. Then we analyzed *why* is the model information encoded, i.e., to solve what task. Finally, we analyzed the relationship between the ML models used with the proposed encodings as well as the purpose in the literature. Based on the findings, as part of our future work, we plan to do a systematic comparative study of different encodings for various ML4CM purposes and use a specific dataset to produce benchmarks for other researchers to use.

References

[1] H. A. Proper and G. Guizzardi, "Modeling for enterprises; let's go to rome via rime," *hand*, vol. 1, p. 3, 2022.

[2] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[3] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Domobot: a bot for automated and interactive domain modelling," in *Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings*, 2020, pp. 1–10.

[4] M. Weyssow, H. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 1071–1089, 2022.

[5] L. Burgueno, J. Cabot, S. Li, and S. Gérard, "A generic lstm neural network architecture to infer heterogeneous model transformations," *Softw. Syst. Model.*, vol. 21, no. 1, pp. 139–156, 2022.

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

[7] J. A. H. López, R. Rubei, J. S. Cuadrado, and D. Di Ruscio, "Machine learning methods for model classification: a comparative study," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, 2022, pp. 165–175.

[8] A. Jacobsen, R. de Miranda Azevedo, N. Juty, D. Batista, S. Coles, R. Cornet, M. Courtot, M. Crosas, M. Dumontier, C. T. Evelo *et al.*, "Fair principles: interpretations and implementation considerations," pp. 10–29, 2020.

[9] D. Bork, S. J. Ali, and B. Roelens, "Conceptual modeling and artificial intelligence: A systematic mapping study," *arXiv preprint arXiv:2303.06758*, 2023.

[10] R. Clarisó and J. Cabot, "Applying graph kernels to model-driven engineering problems," in *1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, 2018, pp. 1–5.

[11] B. Kitchenham, S. Charters *et al.*, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[12] Ö. Babur and L. Cleophas, "Using n-grams for the automated clustering of structural models," in *43rd Int. Conf. on Current Trends in Theory and Practice of Computer Science*, 2017, pp. 510–524.

[13] B. K. Sidhu, K. Singh, and N. Sharma, "A machine learning approach to software model refactoring," *International Journal of Computers and Applications*, vol. 44, no. 2, pp. 166–177, 2022.

[14] G. Lin, X. Kang, K. Liao, F. Zhao, and Y. Chen, "Deep graph learning for semi-supervised classification," *Pattern Recognition*, vol. 118, p. 108039, 2021.

[15] J. A. H. López and J. S. Cuadrado, "An efficient and scalable search engine for models," *Softw. Syst. Model.*, vol. 21, no. 5, pp. 1715–1737, 2022.

[16] M. Fumagalli, T. P. Sales, and G. Guizzardi, "Towards automated support for conceptual model diagnosis and repair," in *Advances in Conceptual Modeling: ER 2020 Workshops*. Springer, 2020, pp. 15–25.

[17] A. Khalilipour, F. Bozyigit, C. Utku, and M. Challenger, "Categorization of the models based on structural information extraction and machine learning," in *Proceedings of the INFUS 2022 Conference, Volume 2*, 2022, pp. 173–181.

[18] A. D. P. Lino and A. Rocha, "Automatic evaluation of erd in e-learning environments," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2018, pp. 1–5.

[19] M. Essaidi, A. Osmani, and C. Rouveirol, "Model-driven data warehouse automation: A dependent-concept learning approach," in *Advances and Applications in Model-Driven Engineering*, 2014, pp. 240–267.

[20] M. Fumagalli, T. P. Sales, and G. Guizzardi, "Pattern discovery in conceptual models using frequent itemset mining," in *41st International Conference on Conceptual Modeling*, 2022, pp. 52–62.

[21] J. Yu, M. Gao, Y. Li, Z. Zhang, W. H. Ip, and K. L. Yung, "Workflow performance prediction based on graph structure aware deep attention neural network," *J. Ind. Inf. Integr.*, vol. 27, p. 100337, 2022.

[22] D. Bork, S. J. Ali, and G. M. Dinev, "Ai-enhanced hybrid decision management," *Bus. Inf. Syst. Eng.*, vol. 65, no. 2, pp. 1–21, 2023.

[23] M. H. Osman, M. R. Chaudron, and P. Van Der Putten, "An analysis of machine learning algorithms for condensing reverse engineered class diagrams," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 140–149.

[24] A. Burattin, P. Soffer, D. Fahland, J. Mendling, H. A. Reijers, I. Vanderfeesten, M. Weidlich, and B. Weber, "Who is behind the model? classifying modelers based on pragmatic model features," in *16th Int. Conference on Business Process Management*, 2018, pp. 322–338.

[25] A. Barriga, R. Heldal, L. Iovino, M. Marthinsen, and A. Rutle, "An extensible framework for customizable model repair," in *Proceedings of the 23rd ACM/IEEE International conference on model driven engineering languages and systems*, 2020, pp. 24–34.

[26] F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Automated clustering of metamodel repositories," in *28th Int. Conf. on Advanced Information Systems Engineering*, 2016, pp. 342–358.

[27] A. Adamu, S. M. Abdulrahman, W. M. N. W. Zainoon, and A. Zakari, "Model matching: Prediction of the influence of uml class diagram parameters during similarity assessment using artificial neural network," *Deep Learning Approaches for Spoken and Natural Language Processing*, pp. 97–109, 2021.

[28] A. Elkamel, M. Gzara, and H. Ben-Abdallah, "An uml class recommender system for software design," in *13th International Conference of Computer Systems and Applications*. IEEE, 2016, pp. 1–8.

[29] X. Dolques, M. Huchard, C. Nebut, and P. Reitz, "Learning transformation rules from transformation examples: An approach based on

relational concept analysis," in *14th IEEE Int. Enterprise Distributed Object Computing Conference Workshops*. IEEE, 2010, pp. 27–32.
[30] L. Almonte, S. Pérez-Soler, E. Guerra, I. Cantador, and J. de Lara, "Automating the synthesis of recommender systems for modelling languages," in *Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering*, 2021, pp. 22–35.
[31] M. Eisenberg, H.-P. Pichler, A. Garmendia, and M. Wimmer, "Towards reinforcement learning for in-place model transformations," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2021, pp. 82–88.
[32] J. Di Rocco, C. Di Sipio, D. Di Ruscio, and P. T. Nguyen, "A gnn-based recommender system to assist the specification of metamodels and models," in *ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems*. IEEE, 2021, pp. 70–81.
[33] P. T. Nguyen, D. Di Ruscio, A. Pierantonio, J. Di Rocco, and L. Iovino, "Convolutional neural networks for enhanced classification mechanisms of metamodels," *J. Syst. Softw.*, vol. 172, p. 110860, 2021.
[34] R. Rubei, J. Di Rocco, D. Di Ruscio, P. T. Nguyen, and A. Pierantonio, "A lightweight approach for the automated classification and clustering of metamodels," in *ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2021, pp. 477–482.
[35] P. T. Nguyen, J. Di Rocco, L. Iovino, D. Di Ruscio, and A. Pierantonio, "Evaluation of a machine learning classifier for metamodels," *Softw. Syst. Model.*, vol. 20, no. 6, pp. 1797–1821, 2021.
[36] Ö. Babur, L. Cleophas, T. Verhoeff, and M. van den Brand, "Towards statistical comparison and analysis of models," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2016, pp. 361–367.
[37] Ö. Babur, L. Cleophas, and M. van den Brand, "Hierarchical clustering of metamodels for comparative analysis and visualization," in *European Conference on Modelling Foundations and Applications*, 2016, pp. 3–18.
[38] ——, "Metamodel clone detection with samos," *Journal of Computer Languages*, vol. 51, pp. 57–74, 2019.
[39] V. Borozanov, S. Hacks, and N. Silva, "Using machine learning techniques for evaluating the similarity of enterprise architecture models," in *Int. Conf. on Advanced Information Systems Engineering*, 2019, pp. 563–578.
[40] P. Danenas and T. Skersys, "Exploring natural language processing in model-to-model transformations," *IEEE Access*, vol. 10, pp. 116 942–116 958, 2022.
[41] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An nlp-based architecture for the autocompletion of partial domain models," in *Advanced Information Systems Engineering: 33rd International Conference, CAiSE 2021*. Springer, 2021, pp. 91–106.
[42] M. Goldstein and C. González-Álvarez, "Augmenting modelers with semantic autocompletion of processes," in *Business Process Management Forum: BPM Forum 2021*. Springer, 2021, pp. 20–36.
[43] G. M. Lahijany, M. Ohrndorf, J. Zenkert, M. Fathi, and U. Kelte, "Identibug: Model-driven visualization of bug reports by extracting class diagram excerpts," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 3317–3323.
[44] S. J. Ali, G. Guizzardi, and D. Bork, "Enabling representation learning in ontology-driven conceptual modeling using graph neural networks," in *Int. Conf. on Advanced Information Systems Engineering*, 2023.
[45] J. A. H. López and J. S. Cuadrado, "Towards the characterization of realistic model generators using graph neural networks," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2021, pp. 58–69.
[46] B. Li, T. Liu, Z. Zhao, P. Wang, and X. Du, "Neural bag-of-ngrams," in *AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.
[47] J. A. H. López, J. L. Cánovas Izquierdo, and J. S. Cuadrado, "Modelset: a dataset for machine learning in model-driven engineering," *Softw. Syst. Model.*, pp. 1–20, 2022.