Word Embeddings for Model-Driven Engineering

José Antonio Hernández López
DIS, Universidad de Murcia
Murcia, Spain
joseantonio.hernandez6@um.es

Carlos Durá
DIS, Universidad de Murcia
Murcia, Spain
carlos.durac@um.es

Jesús Sánchez Cuadrado
DIS, Universidad de Murcia
Murcia, Spain
jesusc@um.es

***Abstract*—Model-Driven Engineering practitioners have to deal with the construction of modelling evironments by devising meta-models, grammars, editors, etc. One of the goals of the application of Machine Learning to MDE is to use ML algorithms to assist the MDE expert in these tasks. These algorithms cannot directly receive raw models or meta-models as input, but they typically have to be transformed into a numeric representation, i.e., a vector. In this context, a common approach is to use pre-trained *Word Embeddings*, which define mapping functions that associate words to semantic vectors. However, current word embeddings are trained with general texts and lack the technical words which typically arise in the modelling domain. To tackle this issue, we have collected a corpus of modelling texts from well-known modelling venues, and we have trained two types of word embedding models. The resulting embeddings (named WordE4MDE) are specialised to address ML tasks in the MDE domain. We have performed an extensive evaluation using the Ecore models of the ModelSet dataset and two state-of-the-art word embeddings (GloVe and Word2Vec) as baselines. We show that WordE4MDE outperforms these two baselines in three meta-modelling tasks, namely meta-model classification, meta-model clustering, and meta-model concept recommendation. WordE4MDE embeddings are available at https://github.com/models-lab/worde4mde and can be loaded using standard Python libraries for their use in ML pipelines.**

***Index Terms*—Model-Driven Engineering, Machine Learning, Word Embeddings**

I. Introduction

The use of Machine Learning (ML) to tackle Model-Driven Engineering (MDE) problems has recently become an active research field. In the last years, a number of proposals about this topic have been presented. For instance, ML algorithms have been used to label repositories of models and meta-models [1]–[3], to build recommender systems for software modelling [4], [5], to assess realistic model generators [6], or to build model transformations [7]. A key element in the application of ML to MDE is to choose a suitable representation of the software artefacts [3]. This representation has to be readable by the ML model and it may have a great impact in the performance of the model. In particular, such representation has many times the form of a numeric vector that encode the features of the given artefact.

Word embeddings is an important tool in the ML toolkit since it enables encoding words as vectors. This technique consists in mapping words to low-dimensinal real-valued vectors which encode the semantic information of the input words [8]. This technique has already been used in the context of MDE to recommend UML concepts [9], [10] by using the similarity between vectors, and to classify models and meta-models [3] by embedding a given artefact into a feature vector. A common approach here is to rely on pre-trained word embeddings released by important institutions such as Stanford (GloVe [11]) or Google (Word2Vec [12]). These embeddings have been trained with an extensive corpus of general text coming from sources like Wikipedia, Twitter, or Google News among others. However, an important issue of these embeddings, for their use in MDE, is that they lack the technical words which typically arise in the modelling domain. For instance, these embeddings do not know about concepts like “Petri net” or may consider that the word “State” is related to countries instead of to state machines.

To overcome the aforementioned issue, we have trained two word embeddings with a large corpus of modelling texts and released WordE4MDE (**Word E**mbeddings **for MDE**). More specifically, we have collected a corpus of MDE texts from well-known modelling conferences and journals, and then we have trained a Word2Vec model [12] and a GloVe model [11] over this corpus. To evaluate the usefulness of our embeddings we have used them in three downstream tasks, namely, meta-model classification, meta-model clustering, and meta-model concept recommendation. As baselines, we consider two widely-used general-purpose word embeddings: GloVe vectors trained with Wikipedia and Gigaword and Word2Vec vectors trained with Google News. Through an extensive evaluation using the Ecore models of the ModelSet dataset [13], we show that our embeddings outperform these baselines in these three tasks, thus concluding that WordE4MDE models are a good choice when dealing with ML tasks related to the usage of technical modelling vocabulary. The embeddings are available at https://github.com/models-lab/worde4mde and they can be loaded through the gensim Python library [14] for their use in standard ML pipelines.

Organization. Section II motivates the need for embeddings in MDE. Section III presents the background needed to understand the paper. Section IV explains how the WordE4MDE vectors have been created. In Section V, we report the evaluation of the proposed embeddings in three tasks: meta-model classification, meta-model clustering, and meta-model concept recommendation. Section VI highlights several aspects of the experiments, findings, and limitations of WordE4MDE. Section VII discusses the related work and Sect. VIII concludes.

II. Motivation

In this section, we motivate the need for word embeddings designed specifically for the MDE domain by showing the inability of current pre-trained word embeddings to deal with technical domains which typically arise when applying ML to MDE.

A. Encoding models as vectors

An important task in MDE is to create domain-specific modeling environments. Many times, this task requires dealing with technical domains, like petri nets, state machines, etc. Typically, the MDE developer needs to create meta-models, grammars, editors, etc. to build the modeling environment. One of the goals of the application of ML to MDE is to help in this type of tasks.

In particular, ML tasks related to dealing with meta-models have been proposed in the literature, like meta-model classification [2], clustering [15], and recommendation [4]. An important part of these approaches is how to vectorize the input (meta-)models so that they can be processed by the underlying ML algorithms. A common approach is to represent a model as a bag of words. Fig. 1 shows the bag-of-words representation of a Statechart meta-model. Essentially, the names of the meta-model are extracted into a multiset. Although this representation discards the meta-model structure, the terms in the meta-model still contains enough information to achieve good results in many tasks (e.g., in model classification as it is shown in [3]). Actually, even structural ML models like graph neural networks need to vectorize part of its input (e.g., string attribute values), and therefore this encoding problem also applies to them.

There are roughly two ways of vectorising bags of words. For the sake of concreteness, we will focus on bags of words to encode meta-models (see Fig. 1):

- *TF-IDF vectorisation*. This technique maps a meta-model to a huge vector whose dimension is the size of the vocabulary (i.e., the number of distinct terms in all the considered meta-models). Given a meta-model, the importance of each word of the meta-model with respect to the collection of meta-models (i.e., the dataset) is reflected by the value in the corresponding dimension of the meta-model vector. For instance, in Fig. 1 the term *state* is the most important to characterize the meta-model. The main issue of the TF-IDF vectorisation is the high dimensionality of the produced vectors. This causes an expensive training phase. Futhermore, distance-based ML algorithms that are fed with TF-IDF vectors suffer from the curse of dimensionality.
- *Word embeddings*. This technique maps each individual word to a low-dimensionality vector. A key property is that a word embedding is a learned representation (i.e., there is a ML model which has learned to produce the mapping from words to vectors) in which words that have similar meanings have close vectors. In this way, a common approach is to rely on off-the-shelf, pre-trained word models which have been trained with some large natural language corpus.
  In this way, to vectorise a meta-model, a *pooling of word vectors* is applied. Pooling is a technique typically used in deep learning to reduce the number of features by applying some summarization operator over a set of vectors (e.g., the average is a very common pooling operator). Thus, each one of the words of the meta-model is mapped to a vector, and the meta-model is represented as the average of all the vectors. The dimension of this representation is the word vectors' dimension that is normally much lower than the TF-IDF vectors. An additional advantage of this approach, is that the vectors encode the semantic meaning of the words.

In summary, current state-of-the-art in ML prefers the use of word embeddings due to their lower dimensionality (which fits better with deep learning architectures) and their ability to capture the word semantics (which could improve the generalization capabilities of the ML models).

B. Limitations of current pre-trained word embeddings

State-of-the-art pre-trained word embeddings may not work well when dealing with modelling artefacts (meta-models, grammars, editors, etc.) which target technical domains. This has been shown empirically in [3] for the meta-modelling domain. The main reason is that the terms that compose the bag-of-word representations are very technical and specific, while current pre-trained word embeddings have been trained with general texts. For instance, we can observe the category distribution of the ModelSet dataset [13] in Fig. 2. It shows that the majority of categories are technical and specific of the modelling domain (e.g., *statemachine, petrinet, modelling, class-diagram*, etc). This means that if we want to succesfully use a pre-trained word model, it must be able to "understand" these words. However, we can see that current word embeddings struggle to capture the proper meaning of these words in the MDE context. In particular, let us consider the GloVe embeddings released by Stanford, trained with Wikipedia 2014 and Gigaword 5. If we feed GloVe with the terms in the meta-model of Fig. 1, we can observe the following:

- The term *statechart* does not belong to the vocabulary of the GloVe embeddings. Thus, we cannot obtain its embedding and the final pooling vector will ignore that term.
- The term *state* does belong to the GloVe vocabulary, but the three most similar terms according to this embedding are *federal*, *states*, and *government*. Thus, the meta-modelling semantics are not properly reflected in the *state*'s word embedding.

These two examples clearly show that GloVe embeddings[1] trained with general texts do not generalize well in the technical modelling domain and, hence, may struggle when being used as part of ML models addressing MDE tasks.

[1]Word2Vec embeddings have similar problems. Please refer to our implementation to check this fact or to try additional examples: https://github.com/models-lab/worde4mde

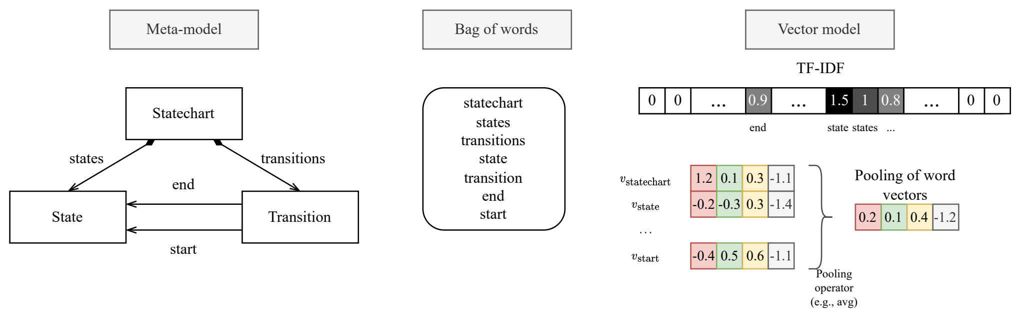

Fig. 1. Bag-of-words vectorisation.

TABLE I
TOP 5 MOST SIMILAR TERMS ACCORDING TO GLOVE TRAINED WITH MDE CORPUS AND GLOVE RELEASED BY STANFORD.

Term	GloVe released by Stanford	GloVe trained with our MDE corpus
state	federal, states, government, department,officials	states, machines, machine, transition, transitions
ecore	bruinsburg, falls-windsor, thongrung, r.winters, osowiec	emf, metamodel, metamodels, gopprr, metametamodel
petri	pasanen, dish, kokko, dishes, laurentius	nets, net, colored, placetransition, stochastic
grammar	vocabulary, syntax, english, school, pronunciation	grammars, contextfree, graph, productions, syntax
atl	roughing, edm, nyr, stl, buf	transformation, transformations, qvt, modeltomodel, mtatl

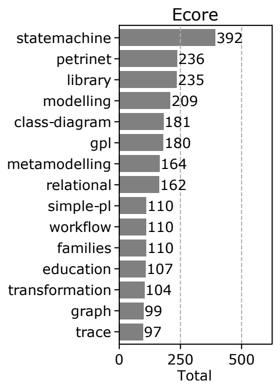

Fig. 2. Top 15 categories of Ecore models in ModelSet. Extracted from [13].

To further stress this fact, Table I shows more modelling terms typically found in the MDE context. The second column shows the top five most similar words given by GloVe, which most of the time fail to capture the intended semantics. In the third column, we show the top five most similar words given by an alternative GloVe model trained by ourselves using a modelling corpus. As it can be observed, the results make more sense and every output term is related to the modelling domain.

III. BACKGROUND

This section presents what word embeddings are and explains more in-depth the Word2Vec model and the GloVe model used in our approach.

A. Word embeddings

Word embeddings are fixed-length, dense, and distributed representations of words which have been proven to be useful in many Natural Language Processing tasks [8]. In particular, they are useful in our scenario to encode modelling artefacts (e.g., meta-models) as vectors. In the literature, several approaches have been proposed to compute such vectors from a given corpus (e.g., one-hot encoding, SVD-based techniques, etc.). However, the ones that are scalable and provide the best results are the iteration-based techniques. This family of approaches relies on a neural network and its internal weights are the embeddings of the words. Well-known techniques that belong to this group are Word2Vec [12] and GloVe [11]. These two models will be explained next since they are the ones used to create the WordE4MDE embeddings.

B. Word2Vec

The Word2Vec tenchique proposes two algorithms to learn the word vectors by using a neural network. The first one is called continuous bag-of-words. Here, the neural network is trained to predict a center word from its surrounding context. For instance, given the sentence `ATL is dynamically typed`, the word `dynamically` is masked and has to be guessed from the context i.e., `ATL is _ typed`. The other one, namely skip-gram, aims to predict the surrounding words by receiving as input the center word. That is, the word `dynamically` is given as input and the neural network is trained to predict the rest of the words. In the remaining of this section, the focus will be on the latter model as it is the one used in WordE4MDE. We have selected skip-gram because it works better with small and specific corpora [16].

Let us set V as the global vocabulary of words. Given a window of size m, that is, a sequence of words composed by a center word, m words on its right, and m words on its left $W = [w_{c-m}, \dots, w_{c-1}, w_c, w_{c+1}, \dots, w_{c+m}]$, the aim of the skip-gram model is to predict $W - \{w_c\}$ using the center word w_c. To do so, the model defines two matrices $\mathcal{V}, \mathcal{U}$ and performs these steps:

1) w_c is transformed into a one-hot vector $x \in \mathbb{R}^{|V|}$.
2) x is mapped to a contextual vector $v_c = \mathcal{V}x \in \mathbb{R}^n$.

3) The contextual vector is used to obtain a score vector $z = \mathcal{U} v_c$.
4) Softmax is applied to z to obtain probabilities $y = \text{softmax}(z)$. The interpretation of $y \in \mathbb{R}^{|V|}$ is the following: the coordinate j represents the probability of the word j (of the vocabulary) to appear in $W - \{w_c\}$.

The columns of $\mathcal{V}$ and the rows of $\mathcal{U}$ are the target embeddings. They are randomly initialized and trained using stochastic gradient descent to minimise the following loss:

$$-\log P_{\mathcal{V},\mathcal{U}}(W - \{w_c\}|w_c) = -\sum_{w \in W - \{w_c\}} \log P_{\mathcal{V},\mathcal{U}}(w|w_c).$$

Finally, we obtain two embeddings (the columns of $\mathcal{V}$ and the rows of $\mathcal{U}$) for each word in the vocabulary. A common approach to obtain the final embedding for each word is to average both embeddings.

C. GloVe

The GloVe model leverages global statistical information by using the global word-word co-occurrence matrix. More specifically, let us denote X as the word-word co-occurrence matrix, where each $X_{i,j}$ is the number of times the word j appear in the context of the word i. The GloVe model minimises a loss function that employs global statistical information of the corpus:

$$\sum_i \sum_j f(X_{i,j}) \left(u_j \cdot v_i - \log X_{i,j} \right)^2 ,$$

where

- $f(\cdot)$ are used to control the weight of rare co-occurrences. It is defined as:
$$f(x) = \begin{cases} (x/x_{max})^{\alpha}, & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{cases}$$
Rare co-occurences are penalized ($x < x_{max}$) and frequent co-occurrences are not overweighted ($x \geq x_{max}$).
- u_j and v_i are the embeddings to be learnt.

The GloVe model consists of minimizing the square error between the dot product of the vectors and the log of the counts $X_{i,j}$ weighted by $f(X_{i,j})$.

IV. Building Word Embeddings for MDE

In this section, we describe how WordE4MDE embeddings have been computed by providing details about the data sources of the corpus, preprocessing steps, corpus statistics, and the traning hyperparameters of the two models that we have considered: skip-gram and GloVe.

A. Data sources

The goal of WordE4MDE is to learn a semantic representation of the technical words which normally appears in the modelling context. Therefore, we need to collect a large amount of texts specifically related to modelling. Our approach has been to rely on well-known modelling, Model-Driven Engineering and Language Engineering venues. The summary of the considered venues is presented in Table II.

TABLE II
Modelling venues considered in the corpus.

Venue	Type	Years	#Articles
MoDELS	Conference	1998-2022	767
MoDELS Companion	Workshops	1998-2022	847
SoSyM	Journal	2002-2022	1039
ER	Conference	1992-2022	1628
ECMFA	Conference	2005-2022	373
ICMT	Conference	2008-2018	182
SLE	Conference	2008-2022	313
ICGT	Conference	1979-2022	379

A complementary set of data sources could be general software engineering venues like IEEE TSE, JSS, IST, ICSE, FSE, etc. Papers from such venues could be filtered by "modelling keywords". However, we have not considered them in this version of WordE4MDE since we could not manage to obtain full institutional access to the articles.

B. Preprocessing

The downloaded articles are in PDF format, therefore we need to convert them to text so that they can be processed by the Word2Vec and GloVe algorithms. The preprocessing steps that we have followed are the following:

1) Article loading. The PDFs were loaded using the pdftotext Python library[2]. We took care that two-column papers are correctly processed, and also handled other details of the text flow like hyphenation.
2) Tokenization into sentences. Each one of the documents was tokenized into sentences by using the nltk sentence tokenizer[3].
3) Sentence normalization. We preprocessed each one of the sentences by removing punctuations and numbers.
4) Word tokenization. We use a nltk word tokenizer to tokenize the sentences. We also lowercase the tokens.

C. Corpus statistics

The processed corpus will be used to train the word embeddings. Table III shows several counts of the processed corpus organised by granularity (number of documents, pages, sentences, and tokens), the number of unique tokens, and the average sentence length. Fig. 3 shows the number of tokens provided by each considered venue to train the word embeddings.

D. Training details

To train the skip-gram (*sgram-mde*) model, we rely on the implementation provided by gensim [14]. The vocabulary is computed by just considering the words which appear, at least, 10 times in the corpus. The dimesion of the word vectors is set to 300 and the window size to 10. We train the embedings during 20 epochs using the negative sampling optimization [12].

[2]https://github.com/jalan/pdftotext
[3]https://www.nltk.org/

TABLE III
SEVERAL STATISTICS OF THE MODELLING CORPUS.

STATISTICS	COUNT
Number of processed pdfs	2 676
Number of pages	87 666
Number of sentences	1 885 647
Number of tokens	39 601 238
Number of unique tokens	455 991
Average sentence length	21.0 $\pm$ 16.99

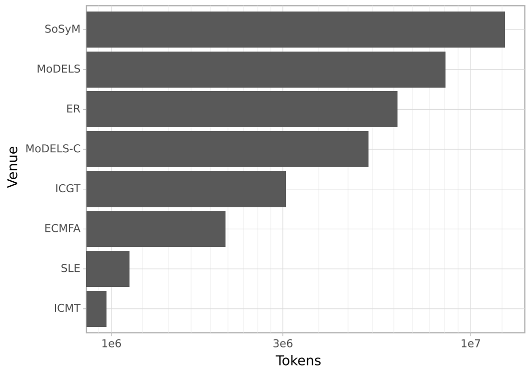

Fig. 3. Number of tokens provided for each venue.

To train the GloVe model (*glove-mde*), we rely on the implementation released by Stanford[4]. We only consider words which appear, at least, 10 times in the corpus. The dimension and window size are also set to 300 and 10 respectively. Finally, x_{max} is set to 10 and α to 0.75. We train the GloVe model during 20 epochs.

The training phase with those hypermarameters and the modelling corpus took about 20 minutes per model using an AMD Ryzen 7 3700X 8-Core Processor with 32 GB of main memory.

V. EVALUATION

The aim of this section is to quantitatively evaluate the quality of our word embeddings by considering three tasks in the context of the meta-modelling domain. The goal is to systematically show the limitation of the current pre-trained embeddings in these tasks, and to compare the performance of WordE4MDE models against current pre-trained models. It is important to note that the three selected tasks (classification, clustering and simple concept recommendation) and the associated ML models are purposely simple, because our goal is to study the effect of the embeddings.

Firstly, we describe the experimental settings by explaining the dataset used in the experiments, the two baselines that we used to compare our approach, and the three tasks considered. Finally, we report the evaluation results.

A. Dataset

The target dataset in our experiments is the ModelSet dataset [13], since it is nowadays the largest labelled dataset in the context of MDE. It contains $\sim$ 10k labelled models. In particular, we take the labelled Ecore models ($\sim$ 5k models). Previous work [3] showed that the presence of duplication could bias the results obtaining optimistic results. Thus, in our analysis, we consider the version of ModelSet without duplication. To filter duplicates, we use the same procedure as [3] that employs an adapted version of the algorithm presented in [17]. From the original dataset, we filter meta-models whose language is not English, meta-models whose category is *unknown* or *dummy*, and meta-models whose category contains less than 10 meta-models. Table IV shows the number of models and categories of the final subset of the ModelSet dataset together with several statistics such as average number of elements, classes, etc.

[4]https://github.com/stanfordnlp/GloVe

TABLE IV
MAIN STATISTICS OF THE SUBSET OF THE MODELSET DATASET CONSIDERED IN THE EXPERIMENTS.

STATISTICS	COUNT
Number of models	2 047
Number of categories	48
Avg. number of elements	143.71
Avg. number of classes	19.12
Avg. number of attributes	12.64
Avg. number of references	21.59
Avg. number of packages	1.34

B. Baselines

To understand the potential improvements brought by the fact that WordE4MDE has been trained on a specific corpus, we compare against two state-of-the-art word embeddings, which has been trained using general texts.

Word2Vec Google News (Word2Vec) Word2vec model released by Google [5]. The model was trained on a part of Google News dataset ($\sim$ 100 billion words). We have considered the embeddings version that contains 300 dimensions to provide fair comparisons with respect to Word2Vec4MDE.

GloVe Wikipedia+Gigaword (GloVe) GloVe model released by Standford [6]. The model was trained on Wikipedia 2014 and Gigaword 5 ($\sim$ 6 billion words). As we do with the previous baseline model, we have considered the embeddings version that contains 300 dimensions.

Therefore, the evaluation considers the two baselines and our two counterpart models (*sgram-mde* and *glove-mde*). Table V compares several characteristics of these four models: training corpus, number of the tokens of the corpus, vocabulary size, and the average ModelSet coverage. The average ModelSet coverage is computed as follows. Firstly, for each meta-model, we calculate the proportion of its tokens that are known by a given word embedding model (i.e., that belongs to its vocabulary). Finally, the average of these proportions is taken. It is important to highlight that, although modelling corpus and the vocabulary are several times smaller than the

[5]https://code.google.com/archive/p/word2vec/
[6]https://nlp.stanford.edu/projects/glove/

baselines, the WordE4MDE vocabulary obtains the highest average coverage.

C. Meta-model classification task

Task. Given a meta-model, this task consists in assigning it a label. As target label, we use the main category of the ModelSet dataset.

Example. Given the meta-model presented in Fig. 1, the system is expected to assign it the label *statemachine*.

Approach. It is illustrated in Fig. 4. Using each word vector model, the meta-models of the dataset are mapped to a vector by computing the average of each vector of the bag-of-word representation. By doing this, each meta-model is represented as a vector that can be used to train an ML classifier. To perform this evaluation, we run (Kernelized) Support Vector Machine classifiers (SVMs) on top of the representations. These classifiers aim to find hyperplanes in the vector space of the representations (if kernels are used, in the transformed vector space) so that we believe that SVMs mainly rely on the information presented in the word embedding model. The hyperparmeters that we have considered are $C \in \{0.01, 0.1, 1, 10, 100\}$ and kernel $\in$ {rbf, linear}. We will report only the results of the best combination of hyperparmeters. The SVM model is thought to solve binary classification problems. Thus, to extend it to a multiclass setting, we use the one-vs-one schema. For each pair of different labels, a SVM is trained. Then, given a new vector, we run all these trained SVMs over this vector, and the label of the majority is the predicted category.

Evaluation metric. To evaluate the performance on the classifiers built on top of the representations, we use the balanced accuracy. In particular, given a category c, its recall is computed as:

$$\text{Recall}_c = \frac{\text{Correctly identified models of the category } c}{\text{Number of samples labelled with } c}.$$

The balanced accuracy is then computed as:

$$\text{balanced accuracy} = \text{AVG}_{c=1}^{C}\text{Recall}_c.$$

We use this metric because ModelSet is not balanced and this measure gives the same importance to each category. To statistically compare the word embeddings models, the $k-$fold procedure is performed and the average of the balanced accuracy of the k folds is used (we set $k = 10$ in our experiments).

D. Meta-model clustering task

Task. Given a non-labelled dataset of models, this task consists in grouping these models in an unsupervised way.

Example. Given the following set of meta-models, $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$ where m_1 models petri-nets, m_2 and m_3 model state machines, and m_4 models relational databases. The system has to output a partition of $\mathcal{M}$ composed by three clusters: $\{m_1\}, \{m_2, m_3\}, \{m_4\}$.

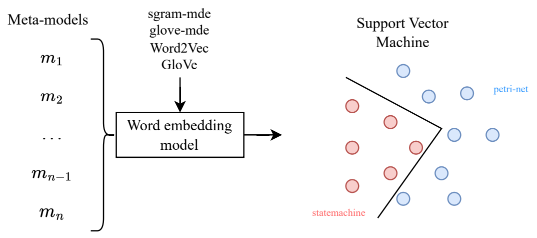

Fig. 4. Meta-model classification approach. For the sake of simplicity, just the SVM trained with the categories *petri-net* and *statemachine* is illustrated.

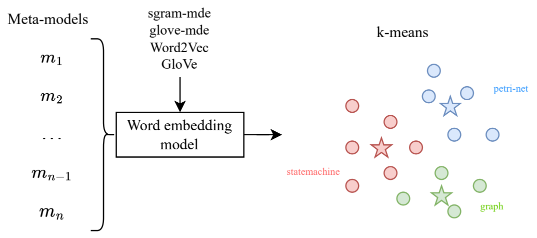

Fig. 5. Meta-model clustering approach. For the sake of simplicity, just the categories *petri-net*, *statemachine*, and *graph* are illustrated.

Approach. This approach is illustrated in Fig. 5. Using the same encoding procedure as the meta-model classification, each meta-model is mapped to a vector by averaging the word vectors of the bag-of-word representation. On top of these vectors, the $k-$means clustering algorithm is run with k equals to the number of distinct categories of the ModelSet dataset.

Evaluation metric. To evaluate whether the ouput clusters make sense, we use as the ground truth the categories of ModelSet. That is, we assess if the output clusters are aligned with the categories. As evaluation metric we use the V-measure score defined as the harmonic mean between homogeneity and completeness. A clustering assignment satisfies the homogeneity property if each cluster contains only members of a single class, and it satisfies the completeness property if all members of a given class are assigned to the same cluster. The homogeneity, completeness, and V-measure scores are between 0.0 and 1.0, and the higher the better [18]. To present the results, we run the $k-$means algorithm ten times with ten distinct seeds and report the average of the V-measure scores.

E. Meta-model concept recommendation task

Task. Given the name of the *container* element, this task consists of recommending potential names for the *contained* elements. In particular, we consider the tasks of recommending *EClassifier* names given the *EPackage* name, recommending *EStructuralFeature* names given the *EClass* name, and recommending *EEnumLiteral* names given the *EEnum* name.

Example. We show three examples (one per recommendation problem):

- Given the EPackage name *Statechart*, the system should recommend meaningful *EClassifiers* names such as *Statechart*, *State*, or *Transition*.

TABLE V
TRAINING CORPUS, VOCABULARY SIZE, AND MODELSET COVERAGE FOR EACH MODEL CONSIDERED IN THE EVALUATION.

MODELS	CORPUS	# TOKENS OF THE CORPUS	VOCABULARY SIZE	AVG. MODELSET COVERAGE
Word2Vec	Google News	∼100B	3 000 000	0.9132
GloVe	Wikipedia + Gigaword	∼6B	400 000	0.9360
sgram-mde/glove-mde	Modelling texts	∼40M	55 519	0.9364

- Given the EClass name *Transition*, the system should recommend *EStructuralFeature* names such as *in*, *out*, *trigger*, etc.
- Given the EEnum name *StateKind*, the system should recommend *EEnumLiteral* names such as *initial*, *final*, or *regular*.

Approach. This approach is illustratred in Fig. 6. Given the name of a *container*, its word vector v_q is obtained using a word vector model. This query vector is transformed into another vector $z_q = W_q v_q$ using the linear transformation W_q. At the same time, each one of the vectors in the word embedding model vocabulary (i.e., the vectors $v_1, \ldots, v_{|V|}$) is transformed through another linear layer W_r obtaining $z_1, \ldots, z_{|V|}$. The k recommendations for the contained elements are the words in the vocabulary whose transformed vectors (z_i) have the highest dot product with respect to the query vector i.e.,

$$\text{argtop\_k}_i\{z_i \cdot z_q | i = 1, \ldots, |V|\}.$$

The matrices W_q and W_r are trained using stochastic gradient descent and training data from the ModelSet dataset, that is, training data in the form of pairs (*container*, *contained elements*). More concretely, let us suppose that z_q (computed using W_q) is the transformed query vector, $Z = \{z_1 \ldots, z_{|V|}\}$ is the set of the transformed vectors of the word models's vocabulary (computed using W_r), and $I = \{r_1, \ldots, r_k\} \subset \{1, \ldots, |V|\}$ are the indices of the ground truth containment elements within the vocabulary. During training, the z_q is compared with each one of the vectors in Z through the dot product, and the softmax activation is applied to map those products to scores within the interval $(0, 1)$:

$$s = \text{softmax}(Z \cdot z_q)$$

Finally, the model is trained (i.e., the matrices W_q and W_r are trained) to maximise the following function:

$$\sum_{i \in I} \log s_{r_i} + \sum_{i=1, i \notin I}^{|V|} \log\left(1 - s_{r_i}\right).$$

The aim of the first element of the sum is to increase the scores of the ground truth names and the aim of the second term is to decrease the scores of the rest of words in the vocabulary. Note that the recommendation model is not complex as we only train linear transformations. Therefore, we believe that the model rely a lot on the input features provided by the word models. We extract the pairs (*container*, *contained elements*) for the input dataset and split them into train set and test set. To provide a fair comparison, we only consider *container* and *contained* concepts that belong to the vocabularies of all the word models.

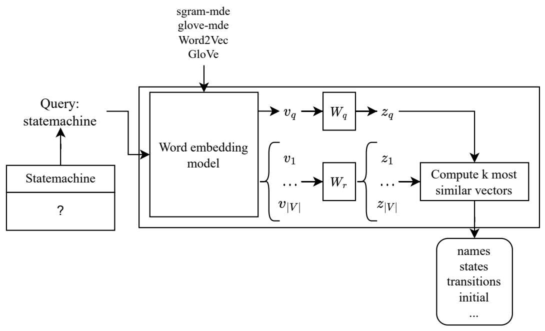

Fig. 6. Recommendation approach in the case of recommending *EStructuralFeature* names.

TABLE VI
META-MODEL CLASSIFICATION RESULTS. AVERAGE BALANCED ACCURACY OF THE TEN FOLDS FOR EACH WORD EMBEDDING MODEL.

	BALANCED ACCURACY
sgram-mde	**0.8190**
glove-mde	0.8034
Word2Vec	0.7809
GloVe	0.7803

Evaluation metric. As evaluation metric, we use Recall@k which is computed as:

$$\text{Recall@}k = \frac{|\text{contained elements} \cap k \text{ recommendations}|}{|\text{containment elements}|}.$$

Finally, we take the average Recall@k score for all container elements in the test set. In our experiments, we consider $k = 10$.

F. Results

Meta-model classification. Table VI shows the balanced accuracy for each considered word embedding model. The sgram-mde model obtains the highest average balanced accuracy. Futhermore, when comparing the performance of the best WordE4MDE model (sgram-mde) with the two baselines, the statistical tests (paired Wilcoxon test plus the Bonferroni correction) reveal statistical differences ($p-$value < 0.05)

TABLE VII
META-MODEL CLUSTERING RESULTS. AVERAGE V-MEASURE SCORE OF THE TEN $k-$MEANS RUNS FOR EACH WORD EMBEDDING MODEL.

	V-MEASURE
sgram-mde	**0.6783**
glove-mde	0.6516
Word2Vec	0.6217
GloVe	0.6211

TABLE VIII
AVERAGE RECALL@10 OF THE TEST SET FOR EACH WORD EMBEDDING MODEL AND RECOMMENDATION TASK.

	RECOMMENDING ECLASSIFIERS	RECOMMENDING ESTRUCTURALFEATURES	RECOMMENDING EENUMLITERALS
sgram-mde	**0.4258**	**0.5626**	**0.3981**
glove-mde	0.4038	0.5241	0.3748
GloVe	0.3494	0.4961	0.3535
Word2Vec	0.2685	0.4740	0.1383

in both comparisons, showing that sgram-mde outperforms Word2Vec and GloVe models.

From these observations, we can conclude the following:

Meta-model classification results. The WordE4MDE embeddings are the best in the meta-model classification task. In particular, the sgram-mde model is the one that performs the best.

Meta-model clustering. Table VII shows the average V-measure score for each considered word embedding model. The sgram-mde model obtains the highest average performance and when comparing such model with the two baselines using statistical tests (paired Wilcoxon test plus the Bonferroni correction), we find statistical differences ($p-$value < 0.01) in both comparisons.

From these observations, we can conclude the following:

Meta-model clustering results. The WordE4MDE embeddings perform the best in the meta-model clustering task. Particularly, sgram-mde is the model that performs the best.

Meta-model concept recommendation. Table VIII shows the average Recall@10 for each considered word embedding model and for each recommendation task. We can observe that the sgram-mde model achieves the best average performance in the three tasks. The results of comparing the best WordE4MDE model with the baselines through statistical tests (paired Wilcoxon test plus the Bonferroni correction) are the following.

1) Recommending *EClassifiers* and recommending *EStructuralFeatures*. Sgram-mde outperforms GloVe ($p-$value< 0.01) and Word2Vec ($p-$value< 0.01).
2) Recommending *EEnumLiterals*. Sgram-mde outperforms Word2Vec ($p-$value< 0.01) but we do not find statistical differences between sgram-mde and GloVe.

From these observations, we can conclude the following:

Meta-model concept recommendation results. WordE4MDE is generally the best choice when dealing with the concept recommendation task. In particular, the sgram-mde model clearly outperforms both baselines in the *EClassifiers* and *EStructuralFeatures* recommendation tasks. In the *EENumLiterals* recommendation task, the best WordE4MDE model (sgram-mde) performs very similar to the best baseline (GloVe).

VI. DISCUSSION

In this section, we discuss more in-depth several aspests of our experiments, limitations, applicability of the released word models, and possible opportunities for future work.

A. *Findings in the experiments*

We have found that the WordE4MDE embeddings (particularly, the sgram-model) clearly outperforms the baselines (GloVe and Word2Vec) in meta-model classification and meta-model clustering. This is caused by the fact that WordE4MDE models are more specialized in MDE than the two baselines. We believe that specialization shows up as 1) a specialized vector space that produces better representations of the words in the context of MDE, and 2) specialized vocabulary. The first point can be checked by considering word similarities (see Table I). The second point can be checked by observing that WordE4MDE vocabulary obtains the highest average coverage (see Table V), and by looking at the words in the ModelSet vocabulary that belong to the WordE4MDE vocabulary but does not belong to the baselines. For instance, Table X shows examples of words that belong to the ModelSet dataset and the sgram-mde model but does not belong to the baselines. These terms provide a lot of information when classifying or clustering a meta-model into a category or group. The absence of important modelling terms in the Glove and Word2Vec vocabularies could explain why they perform worse than WordE4MDE.

WordE4MDE embeddings are generally the best choice when dealing with meta-model concept recommendation. In particular, it is important to highlight its superiority in recommending *EClassifiers* and *EStructuralFeatures* in comparison with the baselines. One possible interpretation of this result is that WordE4MDE embeddings encode very well these two meta-modelling relations. In this context, a natural question arises: what type of meta-modelling relations are encoded in the word vectors? By answering this question, we could throw

TABLE IX
APPLICABILITY OF THE WORD EMBEDDINGS.

PREVIOUS WORKS	TASK	ARCHITECTURE/METHOD	SUMMARY
[3], [13]	Model classification	Several ML models	As input features features for several ML.
[2]	Meta-model classification	Feed-forward NN	As input features to feed the neural network.
[19]	Meta-model classification	CNN	As input features to build the 2D matrix.
[15], [20], [21]	Model/Meta-model clustering	Hierarchical clustering	As input features for the clustering algortihm.
[5]	Model recommendation	LSTM	As input features to feed the neural network.
[9]	Model recommendation	Similarity measure	To complement the general knowledge suggestions.
[22]	Model recommendation	-	The recommendation system used in this paper could be integrated in Droid.

TABLE X
SOME WORDS OF THE MODELSET VOCABULARY THAT BELONGS TO THE WORDE4MDE VOCABULARY BUT NOT TO THE BASELINES VOCABULARIES.

INTERSECTIONS	EXAMPLES
(WordE4MDE − GloVe) ∩ ModelSet	expr, stmt, params, pseudostate statemachine, petrinet, attr
(WordE4MDE − Word2Vec) ∩ ModelSet	expr, uml, ocl, unary, stmt, fsm rhs, lhs, initializer, rdbms

light on how to use word vectors to recommend modelling concepts and which model is the best for recommending concepts in each recommendation task. We leave the answering of this interesting question for future work.

In the experiments, we consider the deduplicated version of the ModelSet dataset. We take that version instead of the original one beacuse the duplication may bias the results by obtaining optimistic results. This is specially the case in classification and recommendation tasks, due to the overlapping between the training and testing sets. We have also run the same experiments considering the original version of ModelSet, and the WordE4MDE embeddings still outperform the baselines leading to the same conclusions but the performance gaps are lower.

B. Limitations

One shortcoming of our pre-trained models is that the same conclusions for Ecore do not hold for UML models. Particularly, we have run several experiments using the ModelSet UML and we found that the WordE4MDE embeddings do not significantly outperform the baselines, but perform very similar to them in the UML classification and clustering tasks. This is because the WordE4MDE embeddings have been trained with technical modelling texts and UML models are many times used to model non-technical domains (e.g., a shop) or technical domains outside modelling (e.g., embedded systems for cars). We leave this limitation to be tackled in future work by increasing the training corpus with more general text or by devising a system that combines two word embedding models.

It is worth mentioning several limitations that are intrinsic to the skip-gram and GloVe models used in this paper. When a word does not belong to the vocabulary, its embedding cannot be obtained. This problem is called the OOV (out-of-vocabulary) problem. We plan to tackle this issue in the future by training and releasing word vectors that employ subword information such as [23]. Another problem of these models is that the output embeddings are not contextualized, that is, they not reflect the fact that the meaning of a word depends on the sentence where it appears. To tackle this problem, state-of-the-art approaches (such as BERT [24], RoBERTa [25], GPT [26], etc.) pre-train transformers architectures [27]. Thus, we leave as future work the training of those architectures with modelling texts and their adaptation to deal with modelling tasks.

C. Applicability

It is important to note that the main aim of this work is not to propose an approach for classifiying, clustering, and recommending concepts but to highlight the limitation of the current pre-trained word embeddings and to release a word model that tackles such limitation. The embeddings presented in this paper are thought to be used as a part of a more complex system. In this paper, the systems built on top of the embeddings are, on purpose, simple and rely a lot on the information already present in such vectors. Thus, we can conclude that the WordE4MDE embeddings contain more useful information for the technical (meta-)modelling domain than the baselines and are a better fit for MDE/ML projects which require embeddings.

In this sense, we have studied how WordE4MDE embeddings could be used to improve systems proposed in previous works. Table IX shows several published systems where the embeddings could be used to improve them. The WordE4MDE embeddings can be used to map bags of words to vectors in order to feed ML models. Thus, it can be used as input features in the classification frameworks proposed in [2], [3], [13], [19], and in the clustering systems proposed in [15], [20], [21]. The proposed embeddings could also be used to improve recommendation systems. For instance, they can be used as input features for the LSTM architecture used in [5] and as a complement to the general knowledge recommendation engine of the system proposed in [9]. Finally, although it is not the main aim of the paper, the recommendation system proposed in this work, used to assess the quality of the embeddings, is novel and could be easily integrated in Droid [22].

VII. RELATED WORK

In this section, we review previous works related the use of ML to address MDE tasks. This section is organised in four parts: word embeddings in MDE, model classification, model clustering, and model recommendation.

A. Word embeddings in MDE

Word embeddings have been used to solve MDE problems in the context of model classification and recommendation. Specifically, the GloVe embeddings are used by Lopez et al. [3] to classify the UML models and Ecore meta-models of ModelSet. They achieve good results in UML but not in Ecore. The embedding proposed in this paper obtains good results in the Ecore domain.

In the context of recommendation, Capuano et al. [10] train a Doc2Vec model over a corpus of reverse-engineered class diagrams and use that model to recommend class diagram concepts. The Doc2Vec model could be a research direction to encode full models into a fixed vector. Thus, we leave the training of the Doc2Vec model with modelling texts and the assessment of its performance in the three meta-modelling tasks as future work.

Burgueño et al. [9] propose a concept recommendation system that interpolates general knowledge recommendations and contextual knowledge recommendations. The general recommendations are provided by using a metric distance over the GloVe vectors, and the contextual recommendations are provided by also using a distance metric but with a GloVe model trained on domain documents. This work and the recommendation system proposed by Burgueño et al. are complementary as the WordE4MDE embeddings could be used as a source of general knowledge.

B. Model classification

The model classification task consists in assigning a label to a given model. For instance, Nguyen et al. [2] tackles a similar classification problem as the one considered in this paper but with a smaller dataset [28] and different labels. Their approach to perform such classification is a simple neural network with a TF-IDF encoding scheme. López et al. [13] propose three model classification tasks: the one that we have tackled in this work, dummy model detection, and tag inference. The dummy model detection task consists in identifying models that contain mostly mock data and were created for testing purposes (*dummy model*). This task was tackled by extracting several counts or features from the models (e.g., number of elements, number of characters, number of dummy names, etc.) and running several classifiers on top of these features. Finally, in the tag inference task, the aim is to assign several tags to a given model. This task is actually a multi-label and multi-classification problem. López et al. train a two-layer neural network using as inputs features TF-IDF vectors. The WordE4MDE embeddings could also be applied to these two tasks as a way to encode the models.

C. Model clustering

Given a dataset of models, the model clustering task consists in generating a partition of this dataset in an unsupervised way. For instance, Basciani et al. [15] use a TD-IDF approach and implement hierarchical clustering with common document similarity measures. Babur et al. propose SAMOS [20], [21], [29] a model analytics platform. One of its features is model clustering. Particularly, the authors propose to use the graph structure of the models by considering $n-$grams to perform clustering and this clustering approach is used to detect meta-model clones [29].

D. Model recommendation

The model recommendation task consists in suggesting modellers relevant elements for the model under construction. In this context, to tackle the task of recommending meta-model concepts, there exist several proposals. For instance, Weyssow et al. [4] employ a transformer model trained in a masked corpus of textual meta-models to recommend relevant meta-modelling concepts. Di Rocco et al. [30] propose MemoRec, a collaborative filtering recommender system to assist in the task of generating meta-models. Graph kernels are used in [31] to provide relevant recommendations to complete the partially specified meta-models.

Apart from meta-models, other types of artifacts have been the main target of several proposed recommender systems. For instance, Di Rocco et al. [5] use a LSTM encoder-decoder architecture to predict the next edit operation. The authors evaluate their approach on a dataset of BPMN models. Burgueño et al. [9] and Capuano et al. [10] use word embeddings to recommend concepts for UML models.

VIII. Conclusion & Future work

In this paper, we have trained two word embeddings with a corpus of modelling texts and released WordE4MDE. Through an extensive evaluation using the ModelSet dataset, we show that these embeddings outperform state-of-the-art general-purpose embeddings in three tasks: meta-model classification, meta-model clustering, and meta-model concept recommendation. WordE4MDE embeddings are available at https://github.com/models-lab/worde4mde and can be easily loaded to tackle modelling tasks.

As future work we plan to tackle the highlighted limitations in the discussion section. Particularly, we plan to increase the training corpus to address the generalization capability of our embeddings in less technical domains (like UML), to study more in-depth which are the meta-modelling relations that are encoded in the embeddings, and to tackle the intrinsic limitations by releasing more word embeddings specialised in the modelling domain. Futhermore, we also want to incorporate the WordE4MDE embeddings in previously proposed approaches to empirically study if their addition improve the performance of those systems.

Acknowledgments

Work supported by grant TED2021-129381B-C22 (SATORI project) funded by MCIN/AEI/10.13039/501100011033.

References

[1] J. A. H. López and J. S. Cuadrado, “Mar: a structure-based search engine for models,” in *Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems*, 2020, pp. 57–67.

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

[3] 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.

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

[5] J. Di Rocco, C. Di Sipio, P. T. Nguyen, D. Di Ruscio, and A. Pierantonio, "Finding with nemo: a recommender system to forecast the next modeling operations," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, 2022, pp. 154–164.

[6] 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.

[7] L. Burgueño, J. Cabot, and S. Gérard, "An lstm-based neural network architecture for model transformations," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 294–299.

[8] F. Almeida and G. Xexéo, "Word embeddings: A survey," *arXiv preprint arXiv:1901.09069*, 2019.

[9] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An nlp-based architecture for the autocompletion of partial domain models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 91–106.

[10] T. Capuano, H. A. Sahraoui, B. Fr'enay, and B. Vanderose, "Learning from code repositories to recommend model classes," *J. Object Technol.*, vol. 21, no. 3, pp. 3:1–11, 2022. [Online]. Available: https://doi.org/10.5381/jot.2022.21.3.a4%7D

[11] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[13] J. A. H. López, J. L. Cánovas Izquierdo, and J. S. Cuadrado, "Modelset: a dataset for machine learning in model-driven engineering," *Software and Systems Modeling*, vol. 21, no. 3, pp. 967–986, 2022.

[14] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[15] F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Automated clustering of metamodel repositories," in *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings 28*. Springer, 2016, pp. 342–358.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[17] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2019, pp. 143–153.

[18] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[19] P. T. Nguyen, D. Di Ruscio, A. Pierantonio, J. Di Rocco, and L. Iovino, "Convolutional neural networks for enhanced classification mechanisms of metamodels," *Journal of Systems and Software*, vol. 172, p. 110860, 2021.

[20] Ö. Babur, L. Cleophas, and M. van den Brand, "Hierarchical clustering of metamodels for comparative analysis and visualization," in *Modelling Foundations and Applications: 12th European Conference, ECMFA 2016, Held as Part of STAF 2016, Vienna, Austria, July 6-7, 2016, Proceedings 12*. Springer, 2016, pp. 3–18.

[21] Ö. Babur, L. Cleophas, and M. Van Den Brand, "Model analytics for feature models: case studies for splot repository," in *Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS*, vol. 2245, 2018, pp. 787–792.

[22] 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.

[23] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[26] A. Radford, "Improving language understanding by generative pre-training," 2018.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[28] Ö. Babur, "A Labeled Ecore Metamodel Dataset for Domain Clustering." [Online]. Available: https://zenodo.org/record/2585456

[29] Ö. Babur, L. Cleophas, and M. van den Brand, "Metamodel clone detection with samos," *Journal of Computer Languages*, vol. 51, pp. 57–74, 2019.

[30] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and A. Pierantonio, "Memorec: a recommender system for assisting modelers in specifying metamodels," *Software and Systems Modeling*, pp. 1–21, 2022.

[31] 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 *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2021, pp. 70–81.