Towards Generating Structurally Realistic Models by Generative Adversarial Networks

Abbas Rahimi*†, Massimo Tisi‡, Shekoufeh Kolahdouz Rahimi§† and Luca Berardinelli*
*Institute of Business Informatics - Software Engineering, Johannes Kepler University, Linz, Austria
†MDSE Research Group, University of Isfahan, Isfahan, Iran
‡IMT Atlantique, LS2N (UMR CNRS 6004), Nantes, France
§School of Arts, University of Roehampton, London, United Kingdom
Email: abbas.rahimi@jku.at, massimo.tisi@imt-atlantique.fr, shekoufeh.rahimi@roehampton.ac.uk, luca.berardinelli@jku.at

***Abstract*—*Context.* Several activities in model-driven engineering (MDE), like model transformation testing, would require the availability of big sets of realistic models. However, the community has failed so far in producing large model repositories, and the lack of freely available industrial models has been raised as one of the most important problems in MDE. Consequently, MDE researchers have developed various tools and methods to generate models using different approaches, such as graph grammar, partitioning, and random generation. However, these tools rarely focus on producing new models, considering their realism.**

***Contribution.* In this work, we utilize generative deep learning, in particular, Generative Adversarial Networks (GANs), to present an approach for generating new structurally realistic models. Built atop the Eclipse Modeling Framework, the proposed tool can produce new artificial models from a metamodel and one big instance model as inputs. Graph-based metrics have been used to evaluate the approach. The preliminary statistical results illustrate that using GANs can be promising for creating new realistic models.**

***Index Terms*—Model generation, MDE, Generative Adversarial Networks, Tool Support**

I. Introduction

Nowadays, it is a common practice to employ models and modeling techniques to facilitate the comprehension of complicated scientific challenges and the design of complex systems in engineering areas. Using models, engineers from various disciplines can gain a holistic perspective on intricate and complex problems at a high level of abstraction. By eliminating unnecessary details, this abstract view gives engineers a better understanding of the challenges in the first step and then offers the possibility of discovering and providing suitable solutions.

As a result, using proper modeling technologies to design and develop software systems can positively impact the quality of such systems. Model Driven Engineering (MDE) considers models the primary artifacts [1]. Subsequently, novel tools and methods have been specifically designed for MDE users, which should be assessed from different standpoints, such as quality, efficiency, and scalability. An outstanding example is the verification and validation of model transformations. Finding appropriate models for these kinds of activities is mostly an intricate and challenging effort, given the diversity and multiplicity of domains [2]. Accordingly, model generation is introduced as a suitable solution to this concern.

The main goal of this work is to leverage Generative Adversarial Networks (GANs) to generate *structurally realistic models* to mitigate the lack of data in MDE. In particular, a model is structurally realistic if it cannot be distinguished from the real ones just by looking at the typed graph structure, ignoring the attribute values [3].

We want to exploit the capability of GANs to learn from a small input dataset and produce a large and diverse set of realistic models.

We are especially interested in generating models for the following purposes:

- *Testing model transformations:* One of the essential pillars of MDE is model transformation. The usual way to perform a model transformation is to prepare a transformation code or program using a specific transformation language (like ATL[1], QVT[2], and ETL[3]). Programming and testing the correctness and performance of such programs is a tedious and time-consuming task because their programming environment often lacks advanced features, like debugging capabilities, required to facilitate their development [1]. Therefore, it is inevitable and necessary to ensure the accuracy of the transformation's execution through testing and evaluation before publishing it. To this end, a data set containing a large number of models is required, while the number of models that can be used for testing is either small or very hard to collect, depending on the respective domains.
- *Feeding neural networks (NN) for MDE:* Machine learning techniques can be used to address various challenging problems that are difficult for humans but relatively straightforward for computers. This has become more influential due to the advancements in hardware over the last few decades and the growth in data storage capacity. Therefore, MDE researchers strive to employ these techniques and address the open challenges in their field in light of the significant outcomes yielded by ML. Consequently, they must provide

[1]https://www.eclipse.org/atl
[2]https://projects.eclipse.org/projects/modeling.mmt.qvt-oml
[3]https://www.eclipse.org/epsilon/doc/etl

a proper model repository or dataset for feeding the neural networks to proceed to their aims.

Among different MDE technologies [4], we chose the Eclipse Modeling Framework (EMF)[4] as the foundational MDE technology for a proof-of-concept implementation of the approach. Therefore, we aim to encode an input EMF model into the corresponding graph first, then feed it to a NN, which can learn the main patterns and link distribution between the graph's internal elements. Finally, the trained network can be used to create new EMF models. It is worth noting, however, that the proposed tool can be adapted to deal with different MDE frameworks by replacing technology-dependent components like model encoders/decoders.

The main contribution of the paper is an approach for model generation based on GAN. The proof-of-concept implementation first changes the problem scope from the model level to the graph level. It then learns the graph's main structure, generates new graphs based on that, and finally converts the new graphs back to the model space.

As part of this tool, a specific model-to-graph encoder has been developed to convert the model into the corresponding graph by receiving an Ecore metamodel and an instance model that conforms to it. This program goes through the instance model based on the structural features of the metamodel and creates an adjacency matrix representing the graph corresponding to that model. This matrix then has been used as the input of the neural network. A graph-to-model decoder has been designed to convert the graph to the respective model. The decoder can create an instance model representing the given adjacency matrix by receiving the metamodel structural information and the generated graph's adjacency matrix.

The rest of the paper is organized as follows. Section II discusses the basic concepts used in this research. In Section III, we investigate the related works. Section IV introduces our proposed approach, including the model-to-graph encoder, the neural network that has been used, and the graph-to-model decoder. We illustrate the evaluation results and the initial findings of this research in Section V. In Section VI, we explore the framework's limitations and provide potential solutions. Finally, Section VII concludes the paper by listing potential improvements, new challenges, and future research directions.

II. Background

This section explains the main concepts utilized in this research: model-driven engineering, generative deep learning, and graphs.

A. Model-driven Engineering

Model-driven software engineering is a technique for applying modeling's advantages to software engineering activities. "Model" is considered an important artifact of software engineering in MDE. MDE strongly emphasizes disregarding the detail and leads software engineers' focus away from low-level perspectives like programming code toward higher-level views, such as the models. Due to this change in perspective, challenges are easier to comprehend, and the engineer can focus on discovering the potential solutions [1].

Model transformation is one of the most fruitful facilities MDE offers software engineers [1]. This capability is crucial because, using that, the executable code in various programming languages and platforms can be generated automatically from the models (if they have acceptable quality and are enriched with constraints). MDE's procedures will take longer than common software engineering techniques, but overall, software projects can be done more quickly, easily, and error-free in the long term due to the reusability of created artifacts.

B. Generative Neural Networks

A generative neural network is a powerful method for learning data distribution using unsupervised learning. It has achieved tremendous success in just a few years. All generative models aim to learn the real distribution of the training dataset to generate new data points by sampling. In the following, we will describe more about GANs.

Ian Goodfellow presented *Generative Adversarial Network* in 2014 as an approach for generative networks by deep learning techniques [5]. In the generative network, which is a type of unsupervised deep learning (DL), patterns or rules are automatically discovered and learned from input data. The foundation of these networks is based on game theory and Nash equilibrium.

GANs are a novel way to train a generative model that consists of two separate neural networks (Figure 1). These two networks are the generator, trained to generate new samples, and the discriminator, which tries to classify the samples as either real (i.e., come from the existing dataset) or fake (i.e., generated by the generator). Both networks work against each other in a zero-sum game until the discriminator is deceived in its predictions in half of the cases. In other words, it means the generator has gained the ability to produce believable and very similar samples of the original dataset [6]. After the training phase of these two neural networks, the discriminator can be used as a recognizer in the trained domain, and the generator can be used to produce new realistic samples.

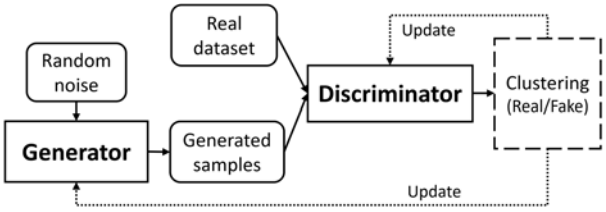

Fig. 1. Generic architecture of Generative Adversarial Networks

C. Graphs

A graph is a visual representation of the relationships (edges) between entities (nodes) in a certain domain. From the

[4]https://eclipse.dev/modeling/emf/

perspective of the nodes' and edges' types, graphs are classified into two main clusters: *homogeneous* and *heterogeneous*.

- A *homogeneous* graph is one in which each node and each edge serve the same intent, and all nodes and all edges in these graphs are of the same type.
- A *heterogeneous* graph has two or more different types of nodes and edges that present different functions and applications [7].

Following this, an *adjacency matrix* is a matrix that contains information about the number of edges between different vertices of a graph. The content of this matrix in unweighted graphs is zeros and ones; if there is a link between two vertices, the respective entry is 1, otherwise 0. In weighted graphs, this value is the edge's weight.

III. Related Work

In recent years, MDE has become one of the leading development methods in many software fields. Consequently, new tools and techniques have been explicitly designed and developed for MDE, which need to be assessed in terms of quality, efficiency, correctness, and scalability. To this end, preparing test models is a challenging, difficult, and time-consuming prerequisite as real instance models are sometimes either inaccessible to the public or even unavailable when it comes to very specific domains [2]. Therefore, in recent decades, MDE researchers have tried to mitigate this challenge by presenting model generation techniques and tools. These studies mainly utilized three approaches: clustering, grammar graph, and random.

The clustering approach usually classifies variable values and relationships between components. Then, an instance model is produced from each category as a representative of that category [8]. To generate the model based on the graph grammar, the graph rules are extracted from the metamodel, and then models are generated according to these rules [9]. In the random approach, random procedures are used to generate new models [10].

Furthermore, in [11], the authors present a viable approach for creating system test data using constraint-solving techniques. It focuses on automating the process of creating effective test cases by formulating constraints based on program behavior and input requirements. Most recently, in [12], the authors have presented a DL-based framework for generating structurally realistic models. They look at the model as a bunch of edit operations and introduce a framework based on deep autoregressive networks. The proposed approach leverages the power of DL networks to model the dependencies across multiple levels, allowing for the generation of highly structured and realistic models. The experiments conducted by the authors show that the deep autoregressive network outperforms existing generative models in terms of capturing complex dependencies and generating high-quality samples.

Moreover, there is an AI-based assessment approach [3] recently done to check whether model generators are producing realistic models or not. In that work, the authors trained a Graph Neural Network (GNN) to capture real-world graphs' underlying structural patterns and characteristics. They used these characteristics to compare the realism of synthetic models generated by different model generators.

We also investigated well-known NN-based frameworks that have been developed for generating realistic graphs. We considered that the given framework should be able to perform effectively and accurately even with a small set of input data because otherwise, it would be a contradictory task. In other words, we need to choose frameworks that are capable of learning the main data patterns using available data since there are not sufficient instance model repositories to feed neural networks in the MDE context. Despite this, it is noteworthy that a larger dataset can yield improved results.

Amongst existing generative networks, we found GAN-based frameworks as a suitable solution to this. To the best of our knowledge, GANs can proceed in the learning phase, even with a small input dataset (e.g., [13]), producing desired and reasonable outcomes. Some of the open-source projects that have tried to address graph generation challenge using GANs are GraphGAN [14], HeGAN [7], and NetGAN [15]. Compared to the others, the NetGAN framework fits our needs better. Moreover, its source code needs less effort to be adopted and is easy to reuse.

NetGAN introduces the graph generation challenge as learning the distribution of random walks over the given graph. NetGAN is based on a stochastic neural network that is able to generate discrete output samples [15]. In other words, it trains the GANs to learn the random walk distribution of the input graph, and then it can reproduce many parts of the given graph's main patterns without having them specified by engineers. Furthermore, it has demonstrated good generalizability that will help the variety of the forged outputs.

IV. Proposed Approach

In this section, we delve into the details of our proposed solution by discussing our approach from the graph perspective and outlining our framework's architecture. We also provide a detailed explanation of key concrete components of the approach.

Models share a structural similarity with graphs. Given the significance of graphs in various fields, such as social network analysis, traffic networks, and protein molecule structure research, numerous studies have recently employed deep learning techniques to generate new samples. We decided to leverage this similarity and utilize existing solutions and tools designed for graph generation to produce new models.

In the following, we detail the stages of our proposed approach depicted in Figure 2.

1) Model to graph encoding: Models can be transformed into graphs inspired by the model's and the graph's structural similarity. To this end, for each element (object) in the input model, a node in the graph is created, and an edge is added for each relationship between every two elements.

With this idea, we used the PyEcore[5] library and provided a Python code to go through the model elements and create

[5]https://github.com/pyecore/pyecore

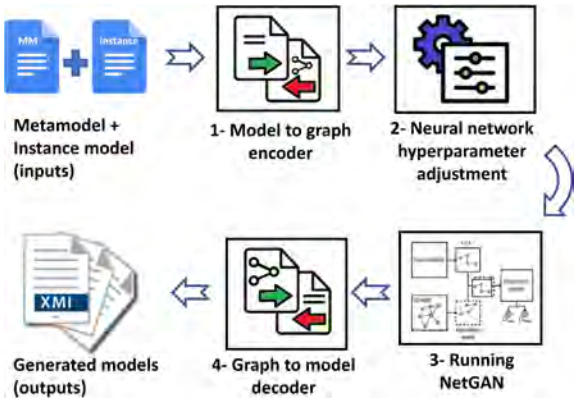

Fig. 2. Workflow of the proposed approach for model generation

the corresponding graph. As shown in Figure 3, the designed encoder receives an Ecore metamodel and one valid instance model as inputs, both serialized in XMI. Then, using the structural information extracted from the metamodel, it navigates through the input instance model and creates an adjacency matrix for it as output. The number of rows and columns in the produced matrix represents the number of existing objects (i.e., instances of EClasses defined in the associated metamodel), while the number of non-zero entries in it reflects the number of relationships (i.e., instances of EReferences defined in the associated metamodel) in the input instance model.

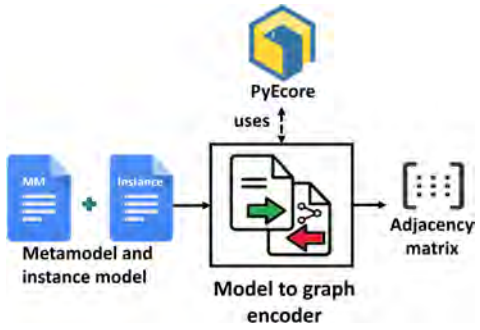

Fig. 3. Model to graph encoder

Input: The input is represented by EMF models, i.e., an Ecore metamodel and a valid instance model. EMF models are all encoded in XMI, i.e., structured text that cannot be used directly as input for neural networks that work on graphs.

2) Adjusting the hyperparameters and running NetGAN: The generator engine and beating heart of the approach proposed in this research is the NetGAN network. The following briefly states the details of the NetGAN framework and how to adjust its hyperparameters.

The GAN framework employed in NetGAN works on random walks, which only considers non-zero entries of the adjacency matrix to work efficiently on the sparse matrix of its input graphs. This capability makes it simple to utilize NetGAN for learning big graphs (for example, a graph with thousands of nodes) [15].

It is important to note that NetGAN has used a Long Short-Term Memory (LSTM) neural network. This type of neural network can effectively remember the main structure and patterns of the input data (an instance model in our case). In conclusion, leveraging the random walks technique and the LSTM architecture by NetGAN makes it a suitable fit for our overall approach.

Before running NetGAN, it is necessary to set and initialize its hyperparameters. The most important parameters are:

- *Random walk (RW) length:* The most influential parameter of NetGAN is the random walk length. This parameter can vary based on the input graph's depth and number of edges. Based on this parameter, NetGAN performs random walks on the input to derive a suitable dataset for training the discriminator. In addition, the generator network also produces fake random walks based on this parameter.
- *Learning rate:* According to the NN model weights update, the learning rate determines how much the model adjusts in response to the calculated error.
- *Stopping criterion:* This determines the permitted percentage of overlap between the generated graph and the original graph.
- *Number of walks synthesized by the generator after training:* After the training phase and before invoking the generation function for creating a new adjacency matrix, the generator requires creating some new random walks. This parameter refers to the number of new random walks. These new random walks are used to build the final output graphs.
- *Approximation of the number of edges in new graphs:* The generator uses this parameter to concatenate new walks and generate a new graph with this number of edges. The number of edges is not exactly the same as this parameter value because, due to the metamodel structure, it might not be possible to add some relations as appear in the new graph.

It is also worth noting that since the quality of NN outputs usually improves by feeding it more data, and since NetGAN creates its dataset by creating a large number of random walks, the larger the input instance model, the better.

3) Graph to model decoding: After training the NetGAN network with the appropriate inputs, its generator NN can be invoked to generate new random walks. These random walks are concatenated to shape an adjacency matrix that represents the new graph. Then, a converter is needed to transform the generated graph into a new instance model to shape the final output.

To do this, we developed a graph-to-model decoder in Python. This code cooperates as a complement to the encoder. The decoder receives the adjacency matrix of the newly created graph from NetGAN and the structural details of the metamodel from the encoder. As shown in Figure 4, the decoder, using the PyEcore library, is able to create the instance model file corresponding to the given graph in XMI format. Moreover, at the end of this stage, each object's features can automatically be initialized via a random value based on their types to make the output more readable.

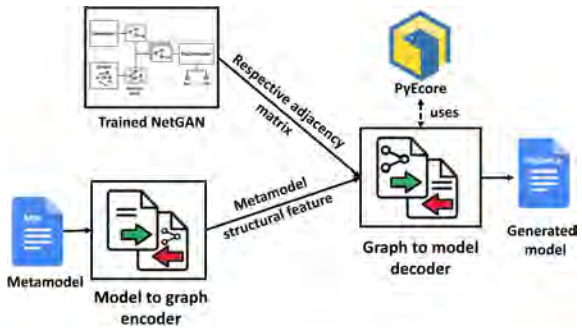

Fig. 4. Graph to model decoder

V. Preliminary Evaluation

By examining the values of an instance model's features, a modeler can quickly distinguish between an automatically generated model and a model created by a human. However, determining a model's authenticity becomes considerably challenging by increasing the level of abstraction by concentrating only on graph structures and removing features' values [3]. However, it is worth noting that applying such an abstraction step comes at the cost of assuming that the feature values are insignificant.

After applying such an abstraction step, the proposed tool is evaluated by analyzing the structural statistics of the generated models. To this end, the graph metrics introduced in [16] are employed. We first calculate these metrics for the real model dataset and the dataset produced by our approach. Subsequently, we compare and evaluate the metrics to assess the level of realism exhibited by the synthetic models.

A. Quantitative metrics used for evaluation

Six evaluation criteria were used, taken from [16], including Node Degree Mean (DM), Node Activity (NA), Node Reference Activity (NRA), Edge Reference Activity (ERA), Multiplex Participation Coefficient (MPC), and Pairwise Multiplexity (PM). Additionally, two more comparative metrics were defined and applied in this study. They are:

- *Absolute distance (AD)*: It is calculated separately for each metric. It is equal to the absolute value of the difference between two graphs in that metric [17]. In the following formula, *M* and *N* refer to two given graphs/models, *s* refers to a metric id, and m_s refers to the value of that metric.

$$AD(M, N, m_s) = |m_s(M) - m_s(N)|$$

- *Combinational distance (CD)*: It is introduced for an in-all comparison of two graphs [17]; This measure combines all the aforementioned metrics and is calculated using the following formula. *k* refers to the number of metrics we have (it's 6 in our case). We defined a weight called w_s for each metric to normalize the value of the metrics. The value of w_s is equal to 1 for metrics whose value is in the range of 0 and 1; otherwise, it is obtained by dividing one by the largest value that has been recorded for that metric.

$$Distance_{comb} = \sum_{1 \le s \le k} W_s.AD_s\big(m_s(M), m_s(N)\big)$$

B. Datasets

As an example showing the proposed approach in action, we used a simplified Car Wash Management metamodel, which is shown in Figure 5. We explain the process of producing, step by step, a new conforming car *instance model* using the proposed approach. The metamodel consists of 5 metaclasses, `CarWash`, `Person`, `Car`, `Service`, and `Bill`. `CarWash` is the top-level model container.

A person owns one or more cars and is responsible for one or more bills. It includes attributes *id*, *name*, and *age*. A car wash station can offer different services at a given *cost* to cars of a specific *color*. The *total cost* of the services is calculated on a bill whose *payment* is performed by cash or credit.

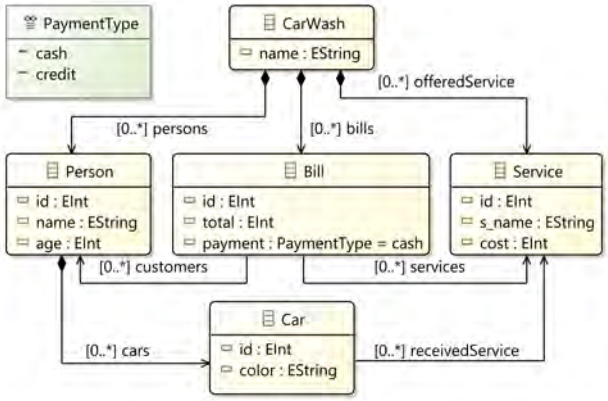

Fig. 5. Car wash management system metamodel.

To gain a dataset crafted by humans, the Car wash metamodel was presented to three modelers, so they created 20 real instance models. Next, the NetGAN was trained five times, utilizing four models among the real models as inputs to generate a dataset of new synthetic ones. After each training iteration, the trained neural network was tasked with generating new models of diverse sizes. Consequently, this process yielded a dataset of 92 generated instance models. It is worth noting that the generator can not generate outputs with more nodes than it sees during the training phase, while it can predict more edges (relationships) than the edges that exist in real models. Figure 6 shows the minimum, maximum, and average number of nodes and edges within the real dataset, training dataset (the four instance models opted from the real dataset), and generated dataset.

Finally, we computed the evaluation metrics for each model and its closest counterpart (i.e., the model with the lowest combinational distance) in terms of similarity in both datasets. These results were subsequently compared to quantitatively assess the similarity between the generated models and their nearest model in the real dataset according to the calculated metrics. The code and dataset can be found at: https://github.com/AbbasRahimi/netgan/tree/Ecore_model_generator.

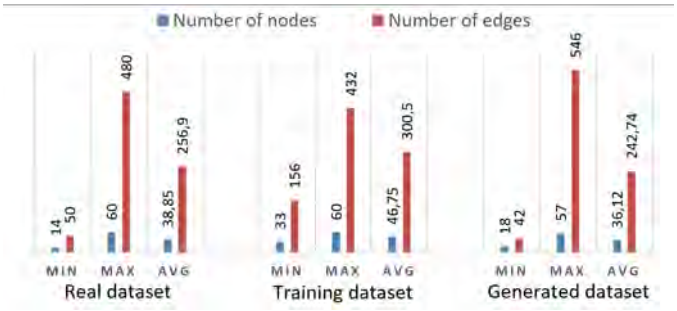

Fig. 6. Instance model's sizes (nodes and edges) in the datasets.

C. Preliminary evaluation results

Upon gathering the required data, we proceeded to compute the CD between each generated model and its nearest counterpart within the real model dataset. This CD is then compared against a threshold limit. To calculate the threshold, first, for each model of the real dataset, we must find the closest model in the same dataset based on the CD [17]. Then, we consider the largest value of the CD difference as the threshold limit. The threshold's value completely depends on the quality, quantity, and proximity of the instance models in the real dataset, so it might differ for different input datasets, even when employing an identical metamodel. Given this circumstance, if the difference is found to be below the threshold, it signifies that the Combinational distance between the generated model and its closest real model is smaller than the distance between the two nearest real models in the dataset. In other words, we can claim that the generated instance model can be considered realistic. On the other hand, if the difference is greater than the threshold, it indicates that the created model has no realistic qualities.

In the following, seven plot-box diagrams, shown in Figures 7 and 8, are used to illustrate the distribution of calculated metric values between the real model dataset (which contains 20 models) and the created model dataset using the proposed tool (which contains 92 models). It is worth noting that all criteria have been normalized to fall within the range of 0 to 1.

Table I presents the comprehensive findings derived from the conducted preliminary evaluation. The first column indicates the name of the sample that is selected as the input, while the second column provides the number of nodes or objects, and the third column is the number of edges or references in that instance model. The fourth column shows the RW length chosen after several trials for the input models. Following the training phase, the network was asked to generate a certain number of new models each time, as indicated in the fifth column. The last column represents the proportion of realistically generated models within each respective instance model set.

It is important to note that the information presented in rows 1 and 2 corresponds to the same input model but with different RW lengths. It means the NN is trained twice using the same instance model. The first training employed an RW length of 6, while the second training utilized an 8-step RW. Despite the fact that a more extensive evaluation is required, the obtained results suggest that an appropriate value for the RW length parameter should be proportional to the number of relationships and the depth of the graph associated with the input model in order to generate suitable outputs.

Notably, by manipulating the value of this parameter alone, a significant alteration in the realistic nature of the produced models was observed.

TABLE I
STATISTICAL RESULT OF THE PRESENTED APPROACH

Input name	#Nodes	#Edges	RW length	#Gen models	Realistic nature
Real432	53	432	6	16	31%
Real432	53	432	8	22	73%
Real352	60	352	6	22	68%
Real262	41	262	6	16	75%
Real156	33	156	6	16	75%

Figure 7 illustrates the performance of the tool's NN across various metrics, namely NA, NRA, ERA, Degree Mean, PM, and MPC. The approach demonstrates commendable performance in metrics NRA, ERA, and Degree Mean, indicating its proficiency in capturing the main patterns of the model. However, it exhibits less proficiency in accurately learning the activity distribution of nodes (NA), particularly when the given model possesses significantly fewer relationships. Furthermore, the comparison of two datasets using Pairwise multiplexity, also depicted in Figure 7, suggests that the tool does not adequately prioritize the generation of binary references. This finding implies that the synthetic models generated by the tool exhibit more diversity than the real dataset.

According to Figure 8, it can be concluded that the current approach, taken as a whole, takes into account graph-based criteria for model generation, leading to the creation of diverse models with a satisfactory level of realism. This conclusion can be reached by comparing the distribution range of the CD criterion between the two categories of models.

VI. DISCUSSION

The model generation challenge is still an open research problem in the MDE context. The presented approach and tool can mitigate this need by leveraging generative deep learning techniques. In the following, we listed the approach highlights and limitations and potential solutions to them.

- The presented approach is a pioneering solution as it employs generative adversarial neural networks for model generation. The primary reason for this choice is that GANs may train with a minimal input dataset and provide desired and plausible results. To put it differently, GANs can prove advantageous for synthesizing novel instance models within stringent scenarios of the lack of input data, a challenge we encounter within the context of MDE.

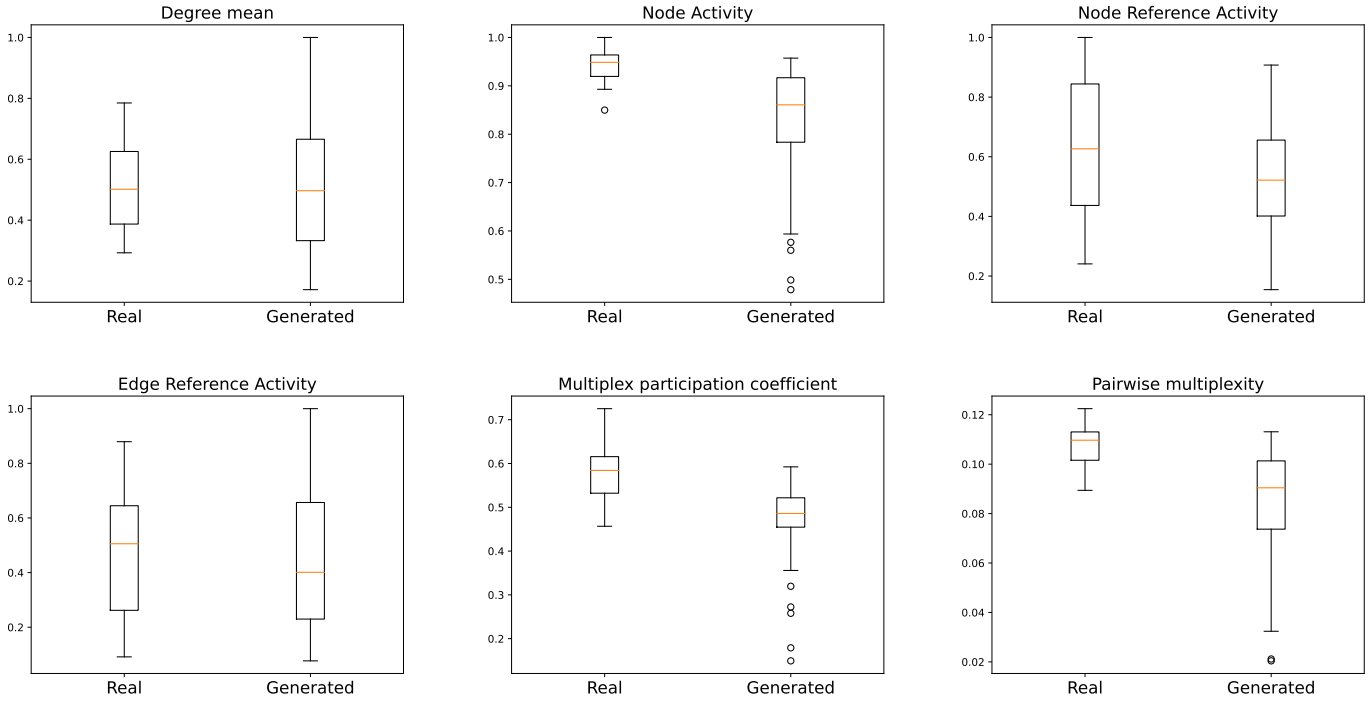

Fig. 7. Plot-box graphs of the evaluation results.

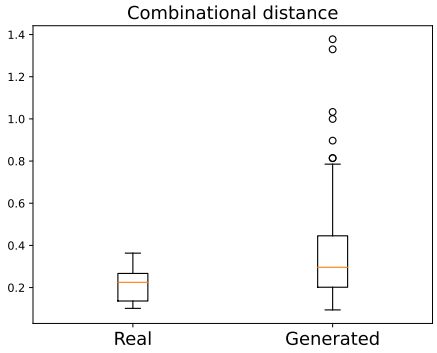

Fig. 8. Combinational distance comparison.

- The provided *M2G encoder* possesses the capability to facilitate the mapping of instance models from the modeling scope to the graph scope. As a result, MDE researchers can leverage this encoder to explore and utilize a wide range of existing facilities and tools that are specifically designed for graphs. This enables them to benefit from the extensive set of resources available for graph analysis and manipulation in their research endeavors.

In the following, we discuss the limitations of our framework and the threats to validity.

- The *NetGAN framework* is specifically designed to operate on homogeneous graphs. Consequently, our proposed approach is limited to generating models that adhere to a metamodel that respects certain specifications. In particular, the input metamodel is allowed to have only one type of relationship between each pair of EClasses. This constraint has to be considered before applying the current approach, as it may not be suitable for generating models with complex or diverse relationship structures. In future work, we aim to mitigate this limitation by either changing the generator NN or improving the encoder and decoder.
- The adjustment of network hyperparameters, such as the learning rate and the number of hidden layers for the generator and the discriminator, can substantially impact both the duration time of the training phase and the quality of the generated outputs. Consequently, one of the major challenges lies in determining the optimal values for these parameters. Finding the right balance is crucial, as selecting inappropriate hyperparameter values may lead to prolonged training times or suboptimal output quality. Therefore, extensive experimentation and fine-tuning are typically required to identify the most suitable hyperparameter values for achieving the desired output quality.
- To trigger the training phase, it is necessary for the size of the input model to exceed a specified minimum value, such as 100 relations. As a result, the presented solution is well-suited for generating instance models that encompass numerous objects and relationships. Therefore, the solution may not be suitable for smaller *instance* models that do not meet the minimum size requirement, and the framework's effectiveness and applicability are primarily optimized for larger-scale *instance* models that exhibit a significant number of entities and relationships.
- During the encoding process of an instance model into a

corresponding graph representation, certain model elements, such as attributes (e.g., EAttributes in EMF) and their corresponding values, may be disregarded and excluded from the NN training phase. It is important to recognize that this overlooked information could potentially hold significant relevance and importance in certain domains.
- The example and the CarWash metamodel are appropriate for an initial experiment and feasibility check of our approach, considering the limitations of the proposed tool. However, further extensive investigation is needed to assess our approach's applicability and reliability.

VII. Conclusion

This research endeavor was conducted to tackle the challenge of model generation within the MDE research field by leveraging recent advancements in generative deep learning. We investigated and assessed the most known varieties of open-source generative neural networks and selected the NetGAN network. Then, a Python code for transforming the EMF models into a graph has been presented, inspired by the significant similarities between the models and graphs in a way that can be fed to the neural networks as input. Consequently, the model-to-graph encoder was developed, which takes an Ecore metamodel and a valid instance model as inputs and generates the adjacency matrix for the respective instance model. Then, the matrix obtained in the previous step is fed into the NetGAN after the network's hyperparameters have been adjusted. It allows NetGAN to produce new graphs following the training phase. Finally, the generated graphs must be transformed back into EMF instance models; therefore, a graph-to-model decoder has been developed that takes an input matrix, converts it to a model, and outputs the result as an EMF model. To assess the quality and degree of realism of the generated models, graph-based criteria were applied to both the real and synthetic datasets. The evaluation results demonstrated that employing generative adversarial networks for model generation yielded the creation of realistic models. These models hold potential value for utilization in different MDE activities and offer opportunities for exploring deep learning-based solutions and approaches for other open challenges within the MDE domain.

In future work, we plan to address the spotted limitations of the approach. We want to assess and exploit the realism of the generated model set by using it as input for concrete MDE tasks, like mutation testing of model transformations. This would allow us to quantitatively compare the generated models' realism and our solution's performance (e.g., by measuring training times) with other existing generators. Finally, we plan to offer our approach as a research solution for industrial use cases proposed by partners in ongoing European and national projects that can benefit from combining MDE and AI techniques.

Acknowledgment

This work was partly funded by the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement number 101007350, and the Federal Ministry of Education, Science and Research of Austria under the TransIT project under the reference number BMBWF-11.102/0033-IV/8/2019.

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering, Cham: Springer International Publishing, 2017.

[2] P. Pietsch, H. S. Yazdi, and U. Kelter, "Generating realistic test models for model processing tools," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 620–623, Nov. 2011. ISSN: 1938-4300.

[3] 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)*, pp. 58–69, Oct. 2021.

[4] A. Iung, J. Carbonell, L. Marchezan, E. Rodrigues, M. Bernardino, F. P. Basso, and B. Medeiros, "Systematic mapping study on domain-specific language development tools," *Empirical Software Engineering*, vol. 25, pp. 4205–4249, 2020.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.

[6] L. Burgueno, J. Cabot, and S. Gerard, "An LSTM-Based Neural Network Architecture for Model Transformations," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, (Munich, Germany), pp. 294–299, IEEE, Sept. 2019.

[7] B. Hu, Y. Fang, and C. Shi, "Adversarial Learning on Heterogeneous Information Networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, (New York, NY, USA), pp. 120–129, Association for Computing Machinery, July 2019.

[8] C. A. G. Pérez and J. Cabot, "Test Data Generation for Model Transformations Combining Partition and Constraint Analysis," vol. 8568, p. 25, July 2014.

[9] S. Sen and B. Baudry, "Mutation-based Model Synthesis in Model Driven Engineering," 2006.

[10] A. Mougenot, A. Darrasse, X. Blanc, and M. Soria, "Uniform Random Generation of Huge Metamodel Instances," in *Model Driven Architecture - Foundations and Applications* (R. F. Paige, A. Hartman, and A. Rensink, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 130–145, Springer, 2009.

[11] G. Soltana, M. Sabetzadeh, and L. C. Briand, "Practical Constraint Solving for Generating System Test Data," *ACM Transactions on Software Engineering and Methodology*, vol. 29, pp. 11:1–11:48, Apr. 2020.

[12] J. A. H. López and J. S. Cuadrado, "Generating structurally realistic models with deep autoregressive networks," *IEEE Transactions on Software Engineering*, pp. 1–16, 2022. Conference Name: IEEE Transactions on Software Engineering.

[13] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (Red Hook, NY, USA), pp. 2234–2242, Curran Associates Inc., Dec. 2016.

[14] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph Representation Learning with Generative Adversarial Nets," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, Nov. 2017.

[15] A. Bojchevski, O. Shchur, D. Z ugner, and S. G unnemann, "NetGAN: Generating Graphs via Random Walks," *arXiv:1803.00816 [cs, stat]*, June 2018. arXiv: 1803.00816.

[16] G. Szárnyas, Z. Kovári, A. Salánki, and D. Varró, "Towards the characterization of realistic models: evaluation of multidisciplinary graph metrics," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, (New York, NY, USA), pp. 87–94, Association for Computing Machinery, Oct. 2016.

[17] O. Semeráth, A. A. Babikian, B. Chen, C. Li, K. Marussy, G. Szárnyas, and D. Varró, "Automated generation of consistent, diverse and structurally realistic graph models," *Software and Systems Modeling*, vol. 20, pp. 1713–1734, Oct. 2021.