2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) | 979-8-3503-0039-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-NIER58687.2023.00008

Towards using Few-Shot Prompt Learning for Automating Model Completion

Meriem Ben Chaaben
DIRO, Université de Montréal
Montréal, Canada
meriem.ben.chaaben@umontreal.ca

Lola Burgueño
University of Malaga and UOC
Malaga and Barcelona, Spain
lolaburgueno@uma.es

Houari Sahraoui
DIRO, Université de Montréal
Montréal, Canada
sahraouh@iro.umontreal.ca

***Abstract*—We propose a simple yet a novel approach to improve completion in domain modeling activities. Our approach exploits the power of large language models by using few-shot prompt learning without the need to train or fine-tune those models with large datasets that are scarce in this field. We implemented our approach and tested it on the completion of static and dynamic domain diagrams. Our initial evaluation shows that such an approach is effective and can be integrated in different ways during the modeling activities.**

***Index Terms*—language models, few-shot learning, prompt learning, domain modeling, model completion.**

I. Introduction and Motivation

Recent developments in deep learning-based language models (LMs) open a world of possibilities to automate and assist software specialists in software development and maintenance tasks. At the implementation level, large code bases allow us to leverage these language models by pre-training them to have good code representations and by fine-tuning them on software engineering specific tasks.

These opportunities are, however, limited when dealing with early software development phases such as analysis and design. Datasets are scarce, and when available, they are not large enough to pre-train or fine-tune deep-learning models. For software modeling activities, several contributions were proposed to circumvent the lack of large datasets. The goal of these research contributions is to recommend domain concepts, their features, and relationships during modeling activities[1].

Di Rocco et al. [1] proposed an approach based on graph kernels that only need a small-size dataset for training. Although the results were promising, the quality of the recommended elements remains too low to be used in real settings. Similarly, Weyssow et al. [2], used such model/metamodel datasets to train an LSTM neural network. Here again, the authors obtained limited results, especially when applied to the iterative construction of a metamodel. Saini et al. [3] proposed a bot to assist modelers with the creation of domain models from requirements expressed in natural language using a combination of NLP techniques. Although their results show that the bot can be useful, they did not exploit LLM, which we believe will have a positive impact in predicting new modeling elements. From another perspective, Capuano et al. [4], exploited the available large code bases to reverse-engineering a set of models to train a RoBERTa language model. The results are acceptable, but the reverse-engineered models on which the authors had to rely for training led to suggestions that reflect implementation aspects rather than the modeled domains. With the goal of exploiting knowledge captured in general and specific natural-language documents, Burgueño et al. [5] used these documents to train language models to suggest model completions. We believe that exploiting natural-language sources is a good idea to overcome the scarcity of the data to exploit deep-learning to assist in modeling activities. However, this work requires to train a language model from scratch for each specific domain, which remains a challenging problem in many scenarios. There are existing models that have been used for different applications such as code completion. An example of these is CoPilot [6], which is specialized in generating and completing code by providing suggestions and auto-completing phrases. This highlights the versatility of language generation models and their potential to be applied in various tasks.

In this paper, we propose the novel idea of exploiting powerful left-to-right LLMs with the aim of completing domain models instead of code.

To this end, we use few-shot prompt learning, which allows us to exploit these LLMs without having to train or fine-tune them on a specific domain or task.

For example, if the goal is to use a LLM to generate the name of the capital of a given country, we need to prepare a prompt where we first provide the LM a description on how to reply to our queries, then we add relevant labeled samples such as two countries and their corresponding capitals (two shots). Finally, we give the country for which we want the LM to provide its capital, i.e., autocomplete. Figure 1 gives an overview on how to use LLM with prompt learning applied to this example of countries and capitals. Note that apart from being an autocomplete engine, these language models are also pattern matching and pattern generation engines. This is why we need to provide not only information about the task to perform but also the pattern that we want them to replicate. As our example shows in Figure 1, the generated text could include further information. For instance, `Japan => Tokyo`

[1]Note that "deep-learning/language model" and "software model" do not refer to the same type of model. To avoid confusion, in this work, each time we refer to a deep-learning or language model (LM), we always refer to it as such, and never as "model" alone.

2832-7632/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-NIER58687.2023.00008

has also been generated, which is not of our interest. To deal with this behaviour, properly transforming queries into prompts and results into modeling elements is an essential part of the approach.

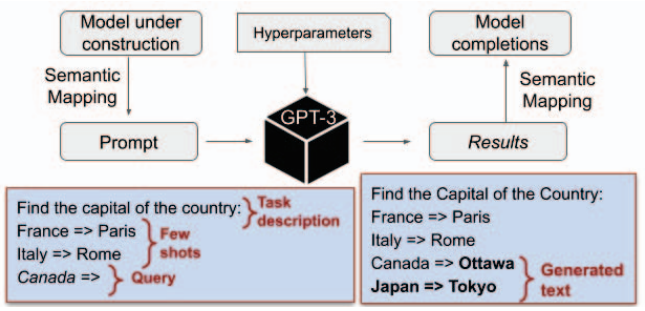

Fig. 1. Explanatory example

We specifically take advantage of GPT-3, which is one of the most powerful LM—it contains 175 billion parameters [7]—, to represent most of the existing general concepts to support software specialists when modeling. To adapt the semantics of the general concepts represented in GPT-3 to the semantics of modeling formalisms, we use two semantic mappings: one that takes the model under construction (i.e., the input to our system) and builds the prompt, and another one that obtains model completion suggestions from the text produced by the language model. These semantic mappings rely heavily on the targeted modeling formalisms.

In this paper, we illustrate our approach with two examples coming from two categories of modeling languages: static models, i.e., UML class diagrams; and dynamic models, i.e., UML activity diagrams. We propose an initial implementation of this idea and a preliminary evaluation.

II. Prompt-Learning for Model Completion

A. Approach Overview

The main goal of our approach is to complete a model under-construction (a.k.a. partial model) by suggesting related elements. Given a partial model, we apply a semantic mapping, i.e, we construct a text representation that will serve as input to GPT-3. We query GPT-3, which returns a textual output that follows a certain pattern. Then, we finish by applying another semantic mapping—in particular, a parsing—to the obtained text and extracting relevant model elements by applying suitable text transformations.

B. Static diagram completion

Using static diagrams for domain modeling usually consists of representing the domain entities, their properties or features and their relationships. For example, the UML class diagrams shown in Figure 2 is a partial description of a banking system. The upper part represents the partial domain model that is already defined by the user.

We focus first on how we design our completion system to suggest entities, i.e., new classes. We create the prompt using some existing diagrams of unrelated domains. From

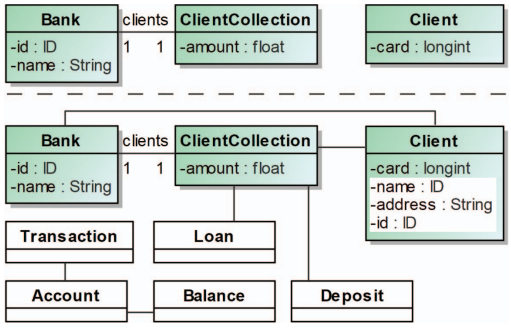

Fig. 2. Example of domain model: Bank class diagram

these diagrams, we extract pairs of related classes and, to follow a certain pattern where we introduce the relationship between two related elements, we represent them between square brackets. Figure 3 shows the few-shots that we provide for our example. Then, we build a query from the partial domain model. To do this, we select between 2 and 4 pairs of related classes, put the classes names in square brackets and add them to the prompt. In Figure 3, under "Generated text" we can observe in bold the text that has been generated for our example. Furthermore, our engine follows a ranking strategy to suggest new elements. We query GPT-3 several times with different prompts, where all the prompts have the same shots but different queries, each query containing a different subset of model elements from the partial model. As a result, we obtain for each prompt a set of suggested concepts. Then, all the obtained concepts from the different prompts are ranked by its frequency from higher to lower. Only those concepts with higher frequency are considered.

Prompt:
Generate related concepts:
hospital: [Nurse, Staff], [Department, Room], [Nurse,patient], [Nurse,department]
reservationSystem: [SpecificFlight, GeneralFlight], [Airport, City], [passenger, plane], [trip, passenger], ...
... (Three more shots)
Bank: [bank, client], [client, clientcollection]
Generated text:
Bank: [bank, client], [client, clientcollection], **[loan, clientcollection], [loan,deposit],[account, balance], [account, transaction]**

Fig. 3. Prompt and generated text for class names and association prediction to complete the Class Diagram of Fig. 2

We apply a string-searching algorithm on the generated text to extract relevant class names and the association that exist between them; we also remove spelling errors and noisy data such as digits, which are usually not part of domain models. In our example, after this step, we obtain that potential missing classes (and associations between them) are `Transaction`, `Balance`, `Deposit`, `Account` and `Loan`. These classes are suggested to the user as shown at the bottom part of Figure 2 with a white color. One can notice that for this

example all the suggestions are closely related to the banking domain that is being modeled.

Given a partial model, to generate prompts for attribute completion, we concatenate the package name and existing class names with their attributes in square brackets, ending with the class for which we are finding potential attributes. Like we do for classes, we use a frequency based ranking function that takes as input all the text generated by GPT-3 for different prompts. Then, we generate attribute suggestions using those concepts which are at the top of the ranking.

Figure 4 illustrates the prompt and resulting text for the class `Client` of Figure 2. We have obtained `name, address` and `id` as potential relevant attributes.

Prompt:
Generate missing attributes for each class in this class diagram:
package company: employee: [id, name, lastName, occupation]; manager: [id, name, department]; company: [name, holding] => employee: [id, name, lastName, occupation, department, experience, revenue]; manager: [id, name, department, team, revenue]; company: [name, holding, address, website]
package bank: bank: [id, name]; clientCollection: [amount]; client: [card]
Generated Text:
package bank: bank: [id, name]; clientCollection: [amount]; client: [card, **name, address, id]**

Fig. 4. Prompt and generated text for attribute completion.

Regarding the suggestion of association names, we design our prompt as follows; from unrelated diagrams, we select pairs of classes that have an association between them. Each pair of classes is used to build a shot by concatenating the name of the classes and the name of the association.

There are different ways of using these prompts—and therefore suggestions—for static diagram completion. We could provide suggestions to the modeler at each iteration of the modeling activity (i.e., one prompt at a time); or after the modeler has completed their diagram to suggest potentially missing elements (i.e., with the combination of the suggestions obtained from many prompts).

C. Dynamic diagram completion

Since structural diagrams do not define sequences, any fragment can be used to generate the prompts to complete a diagram. In the case of dynamic/behavioral diagrams, such as activity diagrams [8], there are strong precedence/sequence constraints to consider (e.g., to represent time), as it can be seen in Figure 5. Hence, to apply prompt learning, we need to define shots and prompts in a way that they preserve those constraints.

In this section, we present how prompt learning can be applied for the completion of activity diagrams. Once again, we need to map the semantics of activity diagrams to a pattern that a LLM such as GPT-3 is able to understand and for which it provides meaningful results. To deal with precedence constraints, we designed our prompts and parameterised them

Mapping	
	start
	end
	=>
	*condition1* / *condition2* => if *condition1* {...} else {...}
	action (as text)

TABLE I
SEMANTIC MAPPING FOR ACTIVITY DIAGRAMS

to predict the next actions in a partial sequence[2]. To build the prompts, we defined simple transformation rules to match the elements of the activity diagram to the appropriate keywords that conform the prompt that we send to GPT-3 as Table I shows. Note that so far, we have only focused on a subset of the activity diagram language.

To illustrate our idea, we introduce the example of an online shopping workflow. Figure 5 shows, in the upper part, the partial activity diagram that is already defined by the user. To create our prompt, we design 3 shots using real activity diagrams extracted from a public repository [9], which have been mapped using the rules described above. Figure 6 represents the prompt for this example and the GPT-3 generated text.

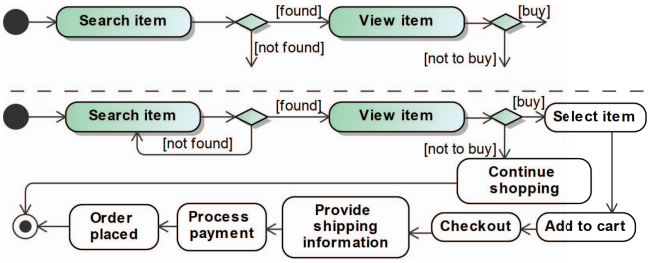

Fig. 5. Example of activity diagram: Online Shopping

Prompt:
Complete the workflow:
Ticket Vending Machine: **start** => request trip info => provide trip info => process trip info => payment request => provide payment Info => process payment => pay with card / pay with cash => **if** with card {authorize card payment} => **end**.
... (two more shots)
Online Shopping: **start** => search item => found / not found => **if** found {view item => buy / not to buy}=>
Generated Text:
if buy {select item => add to cart => checkout=> provide shipping information => process payment => order placed => **end**} **else** {continue shopping => **end**}} **else** {search}.

Fig. 6. Prompt and generated text for the Activity Diagram of Fig. 5

[2]This implies that we need to set the GPT-3 `maximum_number_of_tokens` hyperparameter to a relatively low number—in our experiments it has been set to 50.

After mapping the generated text into model elements, the elements that Figure 5 shows in white represent the completion elements that we obtain. The resulting completions are considered good from two different points of view. From a conceptual point of view, it fits perfectly the domain being modeled; and from a syntactic point of view, the completion suggested comply with the activity diagram syntax.

III. Preliminary evaluation and discussion

The work presented in this paper is an initial attempt to improve the completion of software models. So far, we have evaluated our idea on domain models represented as class diagrams from a public repository.

A. Setup

We followed a manual data sampling methodology, which consisted of selecting 30 domain models from a larger dataset called ModelSet [10]. The goal was to have a dataset with samples of different sizes and containing domain models covering multi-disciplinary domains such as education, finance, entertainment, etc. The manual work was required given the fact that the dataset contained numerous models representing source code and therefore containing low-level implementation details.

Based on our experiments, empirically, we decided to use the GPT-3 model *text-davinci-002* as it consistently produced results that were comparable or superior to other available models. While our approach utilizes a tuning-free prompting—which means the engine generates answers directly without adjusting the parameters of the pre-trained language model taking into account only the provided few shots—we still have to set certain hyper-parameters. One such hyper-parameter is the `temperature`. In general, GPT-3 tends to select words with a higher likelihood of occurring when the temperature is lower. To avoid limiting the search space, we set it between 0.70 and 0.90, as this range provides better results for more creative completions. Another important parameter is the `maximum_number_of_tokens` that can be generated by the model. In our approach, we set this parameter differently depending on the task. For class names and attribute predictions, we set it to 20 as the goal is to generate several words and complete missing concepts, whereas for association name predictions, we set it to 1 as the goal is to generate one precise word.

Furthermore, we have automated the completion process by automatically simulating the behaviour of a modeler using our proof of concept tool. A replication package with the code of our tool and experiments can be found on our Github repository [11].

1) Class names suggestions: To evaluate whether our approach leads to an effective suggestion of new classes, from each model $\mathtt{M}_i$, we took 20% of its elements as the already defined partial diagram $\mathtt{M}'_i$ and we simulated an incremental design process starting from $\mathtt{M}'_i$. First, we performed a first round (R1) of completion suggestions, then we validated the suggested results manually adding to the model under-construction $\mathtt{M}_i{}'$ those which were semantically equivalent to those elements in $\mathtt{M}_i$. The accepted elements were included in the partial model $\mathtt{M}'_i$, resulting in the partial model $\mathtt{M}''_i$. Then, we performed a similar second round (R2) starting from the partial model $\mathtt{M}''_i$. As mentioned before, in each round, we generated three prompts with which we queried GPT-3, each incorporating a varying subset of concepts from the incomplete model. Then, we employed a frequency-based ranking algorithm to determine the final suggested class name.

2) Attributes suggestion: To assess the effectiveness of our approach when suggesting new attributes within a class, we have selected randomly 212 classes from our dataset, have removed 75% of their attributes and have generated attribute completions for them. Note that this means that for classes with three or less attributes, we removed all of them, which was the case for most of the classes. Once we obtained the completion suggestions from our engine, we manually approved those which are either exact matches or semantically equivalent elements to those in the ground-truth model.

3) Association names suggestion: We finally evaluate whether our approach is able to suggest meaningful association names. We extract from our dataset 40 pairs of concepts, where each pair contains the names of two associated classes. Then, we query 3 times our engine to suggest a name for each association, each attempt with the same prompt but a different temperature. Then, we use a frequency-based ranking function to suggest the final association name. We validated manually the output of our engine and approved those which are either exact matches or semantically equivalent to those in the ground-truth model.

B. Results

1) Class names suggestion: As explained previously, we collect results for two successive steps. Table III-B1 summarizes the precision and recall metrics for both steps. We observed that the *Recall* improved from R1 to R2 while the *Precision* decreased slightly. This is due to the fact that the number of correctly suggested elements increases from one round to the next, while the number of incorrect elements increases, too.

	Precision R1	Precision R2	Recall R1	Recall R2
avg	0.57	0.56	0.29	0.45
std	0.26	0.24	0.18	0.25

TABLE II
Results evaluating class names prediction

We also observed that domain models that resulted in the best results (recall between 0.8 to 1) were addressing very common domain/topics used by humans in natural language such as *banking, university* and *library*. Yet, models whose domains contained information that falls further from natural language—such as a model whose package name was `AUni`—resulted into poor results (a recall between 0 and 0.1).

2) Attributes suggestion: For attribute suggestions, we evaluate the recall, defined as the ratio of the ground-truth attributes being found in the Top-N recommended items. In the selected domain models, most classes contain a very limited number of attributes, thus we are only considering the recall metric to check whether we are able to obtain these missing attributes.

The average recall is 0.7 with a standard deviation of 0.4, which can be considered a promising result.

3) Association names suggestion: An interesting metric to evaluate these suggestions is the accuracy, defined as the ratio of correctly predicted association names with respect to the total suggestions. This is, unlike before, we are no longer interested in recognizing the relevant elements, but checking how many times the engine was correct. We have obtained an accuracy of 0.64, which also seems promising.

IV. Discussion

We proposed a novel approach based on few-shot prompt learning to enable large language models to solve completion tasks in modeling activities. We reformulate model completion as a semantic mapping problem that consists, firstly, in transforming modeling formalism elements into meaningful patterns of sequences of tokens to create prompts with learning shots. Then, we exploit the ability of LLMs to complete partial sequences following the specified patterns to recover elements that can be used for the completion. Those elements are transformed into constructs conforming to the modeling language syntax and suggested to the modelers. Although many research contributions were proposed to solve model completion problems, we do believe that none of them can be effectively used in a real setting, because of the resources needed, i.e., large training datasets, and the limited performance they offer. We do believe, however, that our approach can be effective when modeling both static and dynamic diagrams for two main reasons. Firstly, it does not require to pre-train or fine tune language models on specific tasks or domain. Secondly, the used LLMs are trained on a huge volume of data, which makes it generalizable to many domains and different concept natures and relationships.

Although our approach shows promising results, it is still a first attempt and there is room for improvement. Indeed, when defining a prompt, the elements of the already-defined partial diagrams have a great influence on the accuracy of the suggested token. A calibration study is still necessary to determine the boundaries of the provided existing context to have the best suggestions. For example, when we added systematically the package name in the pattern, the results improved considerably, but we cannot determine whether this observation is valid only on the used benchmark. Another consideration that has to be studied is the use of non-natural language elements such as symbols and digits. In our experiments, their existence generated poor results as these elements are rarely present in the data used for training LLM. We believe that a more sophisticated mapping of those elements would considerably improve the results.

Using LLMs proves to be efficient in modeling formalism that rely heavily on natural language identifiers. However, other modeling languages such as Petri nets are definitely difficult to handle as they involve modeling elements that cannot be captured by LLMs.

In our approach, we utilize the advanced capabilities of GPT-3 by OpenAI, a state-of-the-art language model. However, this makes us rely on third-parties, which could limited the availability of the service or deny our access to it in the future. In such case, we should replace GPT-3 with a similar LLM.

Finally, during the evaluation process of our proof of concept, we noticed that, when generating suggestions, most of the time is spent waiting for the responses of GPT-3 after querying its API. With the goal to improve the user experience, a caching system helped to speed up the performance of our implementation by reducing the number of API calls and data cleaning.

V. Conclusion

In this paper we propose an approach to assist modelers in the domain modeling task. Our approach takes advantage of large pre-trained models of natural language through the prompt learning technique. One advantage of our approach is the ability to target different modeling formalisms by defining semantic mapping between the formalism constructs and the natural language concepts. Additionally, the semantic mapping is illustrated with few examples, i.e., few shots, to help the language model find good results for a given prompt. We implemented our approach for both static and dynamic domain models and we report preliminary results for the former models.

The proposed approach, although simple to implement, is powerful and showed promising results. Those results are possibly even better if we consider the fact that the boundaries of a domain are broader than the diagrams we considered as ground truth in our experiments. In fact, some of the suggested elements, although absent in the considered diagrams that we used as ground truth, may be relevant for the domains under consideration. To assess such a claim, we plan to conduct a user study to better assess the correctness of the suggestions, but also the usefulness of the completion for the domain modelers. We also plan to implement a graphical tool for end-users and assess its usability. Finally, we plan to explore how different LLMs can improve the results and even how they can be combined to boost the quality of the suggested model completions.

Acknowledgements

This work has been partially funded by the Natural Sciences and Engineering Research Council of Canada (RGPIN-2019-07168) and by the Spanish Government through the projects IPSCA (PID2021-125527NB-I00) and LOCOSS (PID2020-114615RB-I00).

REFERENCES

[1] J. D. Rocco, C. D. Sipio, D. D. Ruscio, and P. T. Nguyen, "A gnn-based recommender system to assist the specification of metamodels and models," in *24th International Conference on Model Driven Engineering Languages and Systems (MODELS 2022)*. IEEE, 2021, pp. 70–81. [Online]. Available: https://doi.org/10.1109/MODELS50736.2021.00016

[2] M. Weyssow, H. A. 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. [Online]. Available: https://doi.org/10.1007/s10270-022-00975-5

[3] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Automated, interactive, and traceable domain modelling empowered by artificial intelligence," *Software and Systems Modeling*, vol. 21, no. 3, pp. 1015–1045, 2022. [Online]. Available: https://doi.org/10.1007/s10270-021-00942-6

[4] T. Capuano, H. A. Sahraoui, B. Frénay, and B. Vanderose, "Learning from code repositories to recommend model classes," *Journal of Object Technology*, vol. 21, no. 3, pp. 3:1–11, 2022. [Online]. Available: https://doi.org/10.5381/jot.2022.21.3.a4

[5] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An nlp-based architecture for the autocompletion of partial domain models," in *Prof. of the 33rd International Conference on Advanced Information Systems Engineering (CAiSE 2021)*, ser. Lecture Notes in Computer Science, M. L. Rosa, S. W. Sadiq, and E. Teniente, Eds., vol. 12751. Springer, 2021, pp. 91–106. [Online]. Available: https://doi.org/10.1007/978-3-030-79382-1_6

[6] "GitHub Copilot your ai pair programmer," https://github.com/features/copilot/, [Online; accessed 10-January-2023].

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[8] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, version 2.1.1*. OMG, 2007. [Online]. Available: http://www.omg.org/technology/documents/formal/uml.htm

[9] "UML activity diagram examples," https://www.uml-diagrams.org/activity-diagrams-examples.html, [Online; accessed 12-October-2022].

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

[11] M. B. Chaaben, L. Burgueño, and H. Sahraoui, 2023. [Online]. Available: http://hdl.handle.net/20.500.12004/1/C/ICSE/2023/001