MDE in the era of Generative AI

Ahmed ALAOUI MDAGHRI[1], Meriem OUEDERNI[2], and Lotfi CHAARI[2]

[1] University of Nantes, Nantes, France
ahmed.alaoui-mdaghri@etu.univ-nantes.fr
[2] Toulouse INP, University of Toulouse, IRIT, France
firstname.lastname@irit.fr

Abstract. Domain-Specific Languages (DSLs) play a vital role in software development, enabling the concise expression of domain-specific concepts and requirements. In this study, we propose a novel approach leveraging Large Language Models (LLMs) to assist the DSLs modelling starting from natural language description. Our solution is a proof of concept where Model Driven Engineering (MDE) is revisited taking advantage from the power of generative AI. Starting from human friendly description and domain modelling language document type, LLM-based system extracts relevant domain knowledge and builds the corresponding DSL model. Such a model is then validated through an iterative process. We applied our proposal to several case studies from different application domains including software engineering, healthcare, and finance. Furthermore, we consider a wide range of existing LLMs usually adapted for code generation. We also study the effectiveness of our solution through multi-criteria experimental evaluation. Lastly, the results demonstrate the feasibility and efficiency of our LLM driven MDE for DSL development, and then advancing domain-specific modelling practices. By doing so, we would enable the developer to save time and effort for further tasks such as functional properties' verification. A demo as well as a web application for our developed solution are available online via the following link `https://alaouimdaghriahmed.github.io/demo-ecore-gen/`.

Keywords: OMG MDA · MDE · DSL · LLM · Generative AI

1 Introduction

Generative AI (GenAI) is revolutionizing our everyday personal and professional life. It uses sophisticated algorithms to understand contextual knowledge, grammar, and style in order to produce coherent and meaningful output. This is done based on patterns and examples that Gen AI has been trained on. Here, we are particularly interested in Large Language Models (LLMs) which are specific models of GenAI. They rely on prompt engineering and are trained on considerable amounts of text data and learn their statistical properties. In such a context, we focus on the intrinsic link between Software Engineering (SE) and GenAI, and particularly LLMs driven engineering. While the application of GenAI recognize several good impacts, there still many open issues to be studied when developing new software systems. For instance, it is well recognised to achieve general-purpose languages generation using LLMs. However, LLMs are still not well adapted for generating Domain Specific Languages (DSLs).

DSLs are languages designed for specific application domains. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages (GPS, *e.g.* Java, Python, C, etc.) in their domain of application. DSL development is hard, requiring both domain knowledge and language development expertise. However few people could have both.

DSLs can be well developed following Model Driven Engineering (MDE from OMG MDA[3]) methodology. By doing so, both developers and domain experts focus on abstract models rather than actual system code. Here domain experts are the future users and are not usually programmers. Both language expert and domain expert cooperate together for developing a DSL following MDE techniques. Such cooperation enables better understanding of each other, ambiguity resolution and bugs could be detected at early development stage. Hence, this would ease DSL building and also increase software reliability.

The abstract syntax of a DSL is typically defined by a meta-model, which serves as the foundation for establishing the language's structure and rules. To achieve this, a modeling language like Ecore (based on EMOF standard[4]) is suitable and is tool supported within the Eclipse Modeling Framework (EMF).

In the era of LLMs, domain experts find themselves equipped with a potent tool for expression, *i.e.* natural language. Thus, applying LLMs would bridge the gap between both domain and developer experts. However, generating a meta-model from a natural language description faces two challenges. First, the intricate and specific syntax of languages like Ecore can be difficult to manage accurately, especially when the LLMs has not been trained on it. Second, even when the syntax is learnt after giving its technical knowledge into prompting step, the resulting artifacts might be erroneous *w.r.t* that syntax, and/or hold semantics ambiguities. The later issue might arise mainly due to the unclear/incomplete description given to the prompt.

In this paper, we explore the application of LLMs (and LLMs agents) for assisting MDE in order to build new DSLs rather than generating general-purpose languages. We suggest a full process including the syntactic verification and its automatic fix. We also propose human interactive method to fix semantics ambiguity. We apply our approach to several real-world use cases and perform analytic study on given results to validate our process. Our ultimate goal is twofold: *i)* define a new systematic LLM driven MDE methodology, and thus *ii)* provide users with good quality "assistance" in order to ease and improve the DSL design and development cycle. The current work is different from classical code-generation oriented LLMs (starCoder and other similar purpose models). Although such LLMs are not heavy to be manipulated (including 1 to 15 billions of parameters), they remain more adapted for GPL code engineering. To achieve our goal, namely, assisting DSL development following OMG MDA principles, we

[3] `http://www.omg.org/mda/`
[4] `https://www.omg.org/spec/MOF/2.4.1/PDF/`

need to use additional APIs and LLMs agents to perform advanced and complex software tasks.

The remainder of the paper is structured as follows. The Section 2 briefly discusses related research work. Next, Section 3 outlines our LLM-Driven MDE process. Section 4 presents all details about tool support as well well conducted experiments. We synthesize in Section 5 several lessons learned from our work. Finally, Section 6 sums up this work and gives some promising perspectives.

2 Related Work

In this section, we review previous research that is closely related to our work. The recent review given in [4] details several existing work in the application of LLMs for code engineering, ranging from general-purpose generation, summarizing, understanding, comment generation, to example recommendation, etc. To the best of our knowledge, there is no similar work addressing MDE engineering using LLMs for modelling new DSLs as recommended by OMG MDA. In the following, we first explore earlier approaches that uses natural language processing (*i.e.* NLP) techniques for DSL generation, highlighting the foundational methods and their contributions. Later, we explore LLMs agents from the literature since we believe that they are potential candidate to achieve DSL modelling following somehow standardized process.

2.1 LLM for DSL generation

Classical NLP methods consisted of extracting domain specific rules from a textual description. For instance, Aurora et al. [1] successfully did that, while enhancing existing rule sets. Other work by Saini et al. [12] developed a bot aimed at extracting domain models from natural language, providing valuable assistance to novice modelers.

With the advance of NLP techniques, it only made sense to exploit LLMs for DSL exploitation tasks. One methodology outlined by Netz et al. [9] involved leveraging the DSL CD4A, written in MontiCore, to provide both natural language prompts and DSL specifications to an LLM to generate a web application. Jha et al.[5] aimed to fix the hallucination part by exploit the dialog capability of LLMs to iteratively steer them to responses that are consistent with our correctness specification. A lot of other work focused on text-to-SQL tasks and achieved good results using fine-tuning, Retrieval Augmented Generation [6] or even In Context-Learning. Sun et al. [13] explored this task in light of few-shot prompting and instruction fine-tuning. Being our base for instruction fine-tuning our models relative to our task. Our work in a similar way tried to combine these concepts by passing relative information (knowledge or description) about our DSL combine with iterative processing to check and fix errors. However, our approach is generic and applies for modelling any DSL.

2.2 LLM Agents

Although LLMs have not yet reached human levels of problem-solving ability, their achievements are still impressive. Many of these models are pretrained on massive datasets and then fine-tuned for specific problem-solving or coding tasks. However, LLMs often face challenges with complex tasks. This is primarily why LLM agents were introduced. LLM agents are all systems that use LLMs as their engine and can perform actions on their environment based on observations. We state here some agents from the literature *w.r.t* their use case :

– Reflection : The idea is to give back the LLM output as an input with a self correct instruction. Madaan et al. [7] introduces Self-Refine, an approach for improving initial outputs from LLMs through iterative feedback and refinement
– Tool Use : Since APIs are being developed left an right to adhere to users needs why not integrate them with agents. Patil et al. [10] addresses the challenge of effectively using tools via API calls with LLMs. Gorilla, a fine-tuned LLaMA-based model, demonstrates improved performance in writing API calls and mitigates hallucination issues commonly encountered when prompting LLMs directly.
– Task Planning : Planning Consists of deconstructing a problem into easier and manageable sub problems. Wei et al. [15] explore how generating a chain of thought improves the ability of LLMs to perform complex reasoning tasks.

In our research, we investigate and integrate multiple agents to address challenges encountered in DSL design. This approach builds on the work of Qian et al. [11], who decompose complex problems into smaller sub-problems that can be tackled by various AI agents. Typically, an LLM is tasked with dividing the problem and managing the outputs from these AI agents.

3 LLM Driven Engineering

Traditional MDE focuses on creating and exploiting domain models, which serve as the primary artifacts of the engineering process. This classical approach emphasizes the systematic use of models as the main drivers of information exchange, system design, and implementation.

LLM Driven Engineering, on the other hand, leverages the capabilities of LLMs to drive the engineering process based on prompting of natural language description and possibly additional inputs. In our proposal, we considered the following LLMs pillars :

– **Natural Language Usage**: LLMs provide a natural language interface for defining and describing system requirements, designs, and implementations. This reduces the barrier to entry, enabling a broader range of stakeholders to participate in the engineering process.

– **Automatic Code Generation**: LLMs can generate code directly from natural language descriptions, bypassing the need for intermediate abstract models. This can accelerate development and reduce the overhead associated with model creation and transformation.
– **Contextual Understanding**: LLMs possess the ability to understand and incorporate context from a wide range of sources, including documentation, prior interactions, and example data. This enhances their ability to generate relevant and contextually appropriate solutions.

3.1 Achieved Goals

Based on the aforementioned LLMs' pillars, our solution is able to empower model-driven engineering. Through innovative approaches and careful design considerations, our solution ensures the following meaningfully advantages:

– **Automation of DSL Generation:** Our solution automates the process of model generation from human given description, reducing the need for manual intervention and speeding up the development cycle. By leveraging LLMs, we empower domain experts to describe their requirements in natural language. In addition, we equip the developer with language model to alleviate the complexity of design task.
– **Automation of LLM output validation:** Thanks to the use of AI agent collaboration, we are able to solve inconsistencies in our LLM output by iterative prompting and passing the errors back to the LLM until the getting correct output.
– **Enhanced Productivity and Efficiency:** By streamlining the DSL generation process, our solution enhances productivity and efficiency in software development, enabling rapid prototyping and iteration of domain-specific models. The automation of validation and refinement tasks reduces manual effort and accelerates the delivery of high-quality DSLs.

3.2 A new DSL Modelling Process

Our LLM driven MDE for DSL development is sketched on Figure 1 where each enumerated step is detailed below. The proposed process aims at generating a meta-model written in Ecore language (formatted *w.r.t* serialised XMI format [3]), and this is performed based on two inputs : a Natural Language Description and an Ecore technical description. The result is validated using a syntactic parser based on agentic reasoning to handle ambiguous or incomplete requests. By doing so, we ensure accuracy and completeness of generated models.

Notice that we use two different LLM agents' families (see left-side and right-side of Figure 1) depending of our usage needs. One main reason for doing so is that our generation module (on the left-side) is fine-tuned after the generation task, which is not the case for the right-side agents. Another reason is relative to the used inputs and tools. The left-side one is equipped with document parsing and web search while the right-side one needs a syntactic validator.

1. **Specific Prompt:** Our solution starts with an input prompt holding two parameters : *i)* the Natural Language Description (NLD) of a future DSL including user requirements and semantics' constraints; and *ii)* document type definition on a specific modelling language which rigorously gives all technical and syntactic rules that must be satisfied by generated output. For illustration, in the current work, this stands for Ecore definition file passed as second prompt parameter. This is needed because LLMs are not initially trained on such kind of modelling languages. Regarding prompting method, we combine many techniques, namely, grammar prompting [14], COT [15] and tool use [16]. Notice that by doing so, we increase prompting quality taking benefit from different methodologies. We illustrate our prompt template on Figure 2a.
2. **Output Generation :** The LLM processes the input prompt to generate the serialized result (in XMI format) by LLM inference. Considering Ecore meta-model as target output, the inference consists in : first, parsing Ecore language markers and extracting relevant concepts, entities, attributes, and relationships, and then build structural model *w.r.t* the Ecore syntax. Notice that, the inference step could use some components needed for complementary tasks, *e.g.* document parsing.
3. **Model Validation :** Initially, a meta-model is "one-shot" generated (see the right arrow "$\longrightarrow$" going from step 1 to step 2) based on both input parameters mentioned in the initial step 1. This model is then parsed using the domain grammar to check syntactic errors. If no errors are found, the validation process needs human intervention for semantics' checking. He/shed will be asked to provide more clarifications on missing/ambiguous semantics' related features. If this is validated then the validated process terminates and then step 4 on Figure 1 could be performed. However, if syntactic errors raise, the algorithm incrementally fixes each of them thanks to LLM agents and following iterative prompting. For each increment (*i.e.* repeating step 3), both raised error and its corresponding meta-model are passed once again into a LLM using the prompt as shown on Figure 2b. The LLM is then prompted to fix detected errors. The meta-model is updated with the suggested corrections, and the verification is restarted. This validation cycle ends up once the meta-model is free from syntactic errors. Regarding the process modelled in Figure 1, we formalize our syntactic validation method in Algorithm 1.

   Regarding semantics' quality and/or precision, this is left to domain expert appreciation. Our current goal is to assist both domain and developer experts. Domain expert could add some precision to be taken into consideration during the iterative prompting to improve and complete the previously generated result. This would be also helpful for our ambiguity resolution as explained further in step 5.
4. **Database Storage for Use Cases :** Validated outputs are stored in a database for further analysis and fine-tuning of the LLM. This repository of use cases facilitates iterative refinement of the generation process, enabling

continuous improvement based on feedback and real-world application scenarios.

5. **Ambiguity Resolution :** In cases where the generated model fails the validation step[5] or encounters unresolved ambiguity (*e.g.* non explicable issue), our solution re-prompt the LLM with the occurred error and possibly additional description from the user, enabling it to revisit and refine its understanding of the domain context. Additionally, access to external tools such as API search calls and documentation enhances the LLM's understanding and resolution of ambiguous requests.
6. **Fine-tuning and model improvements :** After correcting the generate model based on iterative reasoning and additional inputs, the validated results get stored, after gathering enough validated models, we can use them to fine-tune our LLMs.

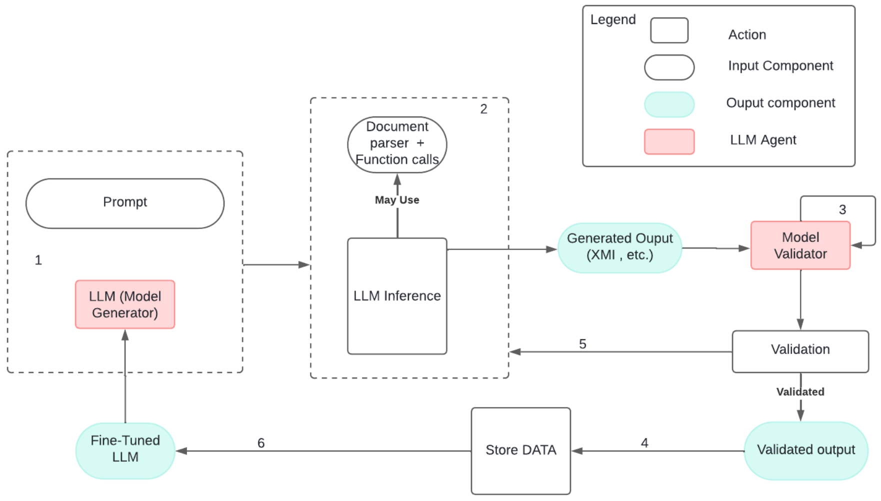


Fig. 1: Design of the proposed solution.

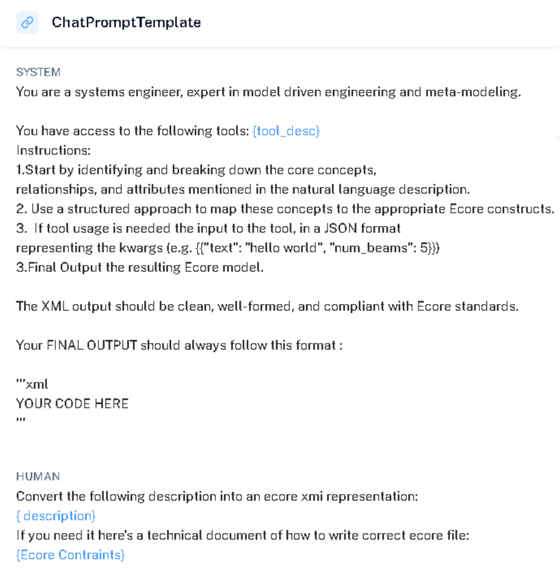
4 Tool Support

This section provides a comprehensive overview of the implementation details of our study, including the used models and the specific use cases to which they were applied. We will delve into the technical aspects of the system. Additionally, we will present the various models selected for this research, elucidating their roles and functionalities within our framework. Lastly, we will describe one use case in detail, highlighting the practical applications and the outcomes observed.

[5] there is a maximum iteration number concluded by learning from most examples

Algorithm 1 Iterative Prompting for Model Validation.

```
Input: Description text (NLD)
Output: Validated meta-model
Procedure:
Generate the meta-model or model from the description text
while True do
    Parse the model using dom_grammar to check for syntactic errors
    if no syntactic errors found then
        break
    else
        Get the list of syntactic errors (from pyecore use)
        for each error in the list do
            Pass the meta-model and the error to the LLM
            Ask the LLM to fix the error
            Update the meta-model with the fix
        end for
    end if
end while
Return Validated meta-model
```



ChatPromptTemplate

SYSTEM
You are a systems engineer, expert in model driven engineering and meta-modeling.

You have access to the following tools: {tool_desc}
Instructions:
1.Start by identifying and breaking down the core concepts,
relationships, and attributes mentioned in the natural language description.
2. Use a structured approach to map these concepts to the appropriate Ecore constructs.
3. If tool usage is needed the input to the tool, in a JSON format
representing the kwargs (e.g. {{"text": "hello world", "num_beams": 5}})
3.Final Output the resulting Ecore model.

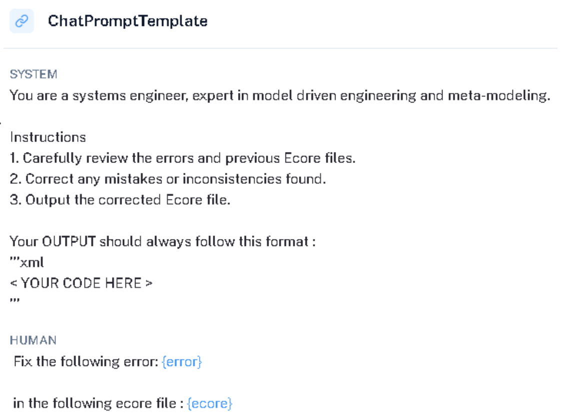
The XML output should be clean, well-formed, and compliant with Ecore standards.

Your FINAL OUTPUT should always follow this format :

'''xml
YOUR CODE HERE
'''

HUMAN
Convert the following description into an ecore xmi representation:
{ description}
If you need it here's a technical document of how to write correct ecore file:
{Ecore Contraints}

(a) Initital prompt template



ChatPromptTemplate

SYSTEM
You are a systems engineer, expert in model driven engineering and meta-modeling.

Instructions
1. Carefully review the errors and previous Ecore files.
2. Correct any mistakes or inconsistencies found.
3. Output the corrected Ecore file.

Your OUTPUT should always follow this format :
'''xml
< YOUR CODE HERE >
'''

HUMAN
Fix the following error: {error}

in the following ecore file : {ecore}

(b) Error fixing prompt template

Fig. 2: Prompt templates.

4.1 Implementation

In order to implement our approach, we have started by studying several existing LLMs. We summarize in Table 1 the most significant ones for our purposes while providing more details on the model weights, if they are open source and how they have been used. Other models were utilized in our study, and they will be discussed subsequently with an analysis of the reasons for not considering them viable options. Notice that both weights and contextual window columns stand for the amount of, respectively, the trainable LLM's parameters, and the tokens we can pass to an LLM without loosing track of its context [8]. Referenced Hugging-Chat-API [6] is an open source project to access the HuggingFace Chat [7] granting access to the latest models in a production environment. Grid5000 [8] is is a large-scale and flexible test-bed for experiment-driven research in all areas of computer science. We used Grid5000 to run lighter models and the Hugging-Chat-API to use already deployed heavier models.

Table 1: Available Large Language Models characteristics.

Name	**Owner**	**Model Weights**	**Inference Endpoint**	**Open-Source**	Context Window
GPT-4 Omni	OpenAI	-	OpenAI API	No	128K
GPT-4 Turbo	OpenAI	8x220 billion parameters	OpenAI API	No	128K
GPT-3.5 Turbo	OpenAI	175 billion parameters	OpenAI API	No	16K
LLaMA3-70B	Meta	70 billion parameters	Grid5000	Yes	8K
LLaMA3-8B	Meta	8 billion parameters	HuggingChat API	Yes	8K
LLaMA2-7B	Meta	7 billion parametes	Grid5000	Yes	32K
Mixtral-8x7B	Mistral AI	8x7 billion parameters	HuggingChat API	Yes	32K
Mistral-7B	Mistral AI	7 billion parameters	Grid5000	Yes	8K
Gemma-7B	Google	7 billion parameters	HuggingChat API	Yes	8K
Gemma-2B	Google	2 billion parameters	Grid5000	Yes	8K
C4AI-Command-r	C4AI	104 billion parameters	HuggingChat API	Yes	128K

Our main use case involved the generation and validation of Ecore models using Pyecore for parsing the generated artifacts. We provided detailed management of these artifacts through several key steps. First, a natural language description was used as a prompt parameter to generate a model using a LLM. Second, a technical Ecore description was employed as a prompt parameter to give the language model better insight into writing syntactically correct Ecore files. Third, we developed a model validator using Pyecore [9] as a library for parsing our produced models and metamodels with Python. The errors identified during this process were utilized in conjunction with iterative prompting (see step 3 in Section 3) as presented in Algorithm 1.

[6] `https://github.com/Soulter/hugging-chat-api`
[7] `https://huggingface.co/chat`
[8] `https://www.grid5000.fr/w/Grid5000:Home`
[9] `https://github.com/pyecore/pyecore`

4.2 Experimentation

We applied our approach to several real-world use cases gathered from the literature. Table 2 cites some of them for illustration. During the experimentation, we collected both the natural language description and the corresponding DSL meta-model. We then applied our approach to generate the Ecore meta-model for each use case. Lastly, we systemically compared our result with the one given by the correspondent reference (mentioned Table 2). We noticed that our generated output is identical to the existing result for all examples.

Table 2: Some Checked use cases.

Use Case	Reference
SimplePDL	`https://eclipse.dev/atl/usecases/SimplePDL2Tina/`
FSM	`http://melange.inria.fr/defining-an-executable-dsl/`
Website Phone friendly	`https://olegoaer.developpez.com/tutos/model/xtext/wdl/`
GemRBACCTX	`https://orbilu.uni.lu/bitstream/10993/22759/1/codaspy2016.pdf`
MontiArc	`https://github.com/MontiCore/montiarc/blob/develop/languages/MontiArc.md`

We now illustrate how our approach does work throughout one use case called SimplePDL [2]. This stands for a DSL which describes software developpement process. Here, we initially provide the LLM prompt with its natural language description as well as the Ecore document type, and we get as an output the corresponding meta-model in Ecore format. Figure 3 shows the graphical diagram of the meta-model.

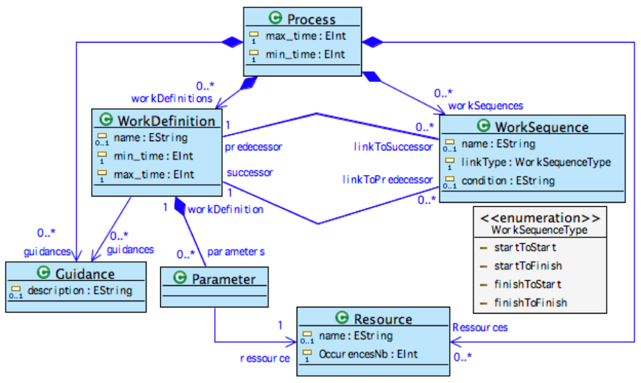


Fig. 3: SimplePDL metamodel.

Later, we proceed as follows. The first one-shot output will be passed to our verification module and checks whether it is correct. For instance, when we applied GPT4-Turbo for this DSL, we found the following error : **"Namespace prefix xsi for type on eClassifiers is not defined, line 3, column 55**

(SIMPLEPDL0.ecore, line 3)". After error fix, the output is re-parsed again and another error is then found, namely, **"Unknown eType in eClassifier"**. Similarly, the erroneous output is fixed and checked again. At this level, no more errors are detected. It concludes that in two iterations we have an error free Ecore file.

5 Learned Lessons

In order to evaluate the proposed method, namely, LLM-driven domain specific modelling, we discuss in this section some analytic studies we carried out for this purpose. We systematically comment on our implementation details, describe usage and validation of our meta-model, and present qualitative results. The analysis presented in the remainder of this paper is stored in a preliminary dataset available online [10].

5.1 Syntactic Error Frequency by Error Category

The syntactic errors generated by different LLMs (mentioned on Figure 5) were categorized into the most frequent types where "Other" category is left for less common errors:

– Invalid comment : this category includes errors related to the use of unsupported tags in XMI.
– Start tag : This category encompasses errors occurring at the beginning of the file.
– Wrong declaration : This category includes errors involving the use of incorrect types or attributes.
– Other : This category comprises miscellaneous errors, including empty files or unsupported syntax.

Figure 4 illustrates the overall occurrence (denoted frequency) of errors categorized as "Invalid Comment," "Other," "Start Tag," and "Wrong Declaration". This plot reveals that "Invalid Comment" is the most common error type, respectively followed by "Other", "Start Tag", and "Wrong Declaration". More precisely, almost half encountered errors belong to the "Invalid Comment" category. This is due to a confusion made by LLMs between XML [3] tags and Ecore serialization (*i.e.* XMI tags). In order to fix detected issues we rely on our iterative prompting method (see Section 5.3).

5.2 Syntactic Error Category Distribution by LLMs

The plot, shown in Figure 5, presents the error category distribution for each LLMs being evaluated. LLMs with very low application frequency do not enable us to generate well-formed Ecore code, *i.e.* no possible parse *w.r.t* standard

[10] `https://huggingface.co/datasets/VeryMadSoul/Errors`

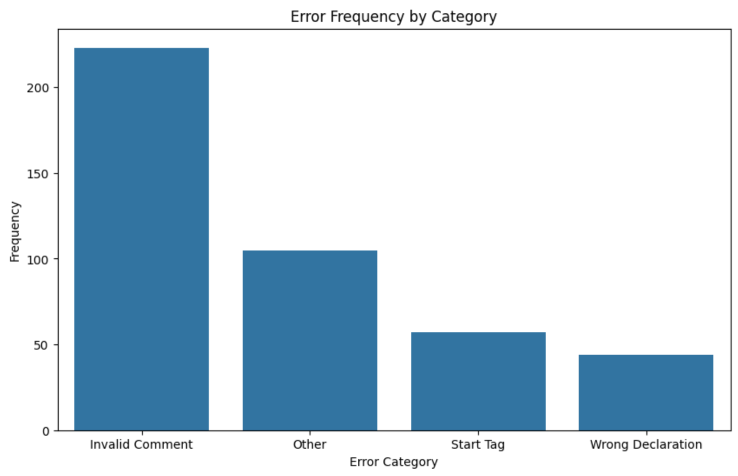


Fig. 4: Error distribution by category.

syntax. In other word, those LLMs would generate code in XML rather than XMI format with the prevalent error being "Start Tag" or "Empty File". From the plot, it is clear that "gpt-4.0", "Llama-3-70B" and "Mixtral-8x7B" models have a high frequency of "Invalid Comment" errors, while the other models exhibit relatively fewer instances of other error categories, such as "Other", "Start Tag", and "Wrong Declaration".

5.3 Distribution of Use Case Resolutions for Each LLM

Table 3 summarizes the distribution of use case resolutions for each LLMs. These statistics were collected based on the resolution rate of one use case (*i.e* SimplePDL) for which we repeated our full process given in Figure 1 40 times. This aims at evaluate the LLMs efficiency. Notice since this task is time and resources consuming, we have been limited to fix the execution repetition to 40 times.

It makes sense to have a 0% correct output for zero-shot context as this work aims to solve exactly that, we also saw that the number of low correct outputs using our prompt for GPT-4o is due to generating comment tags that aren't supported while most of these mistakes would be correct, in 8 cases the model could not resolve the issue. Meta-Llama-3 being the best at generating correct ecore format using only the technical file we provide, still noticing an increase using iterative prompting. Even in the case of Mixtral-8x7B the results tend to improve using iterative prompting

Similar to GTP-4o, the other OPENAI's models tend to perform better at error resolution but not at one shot generation. In terms of the type of error committed, they tend to generated a bit more diverse errors, entailing a further hypothesis, that GPT-4o has seen more XML syntax in its training data.

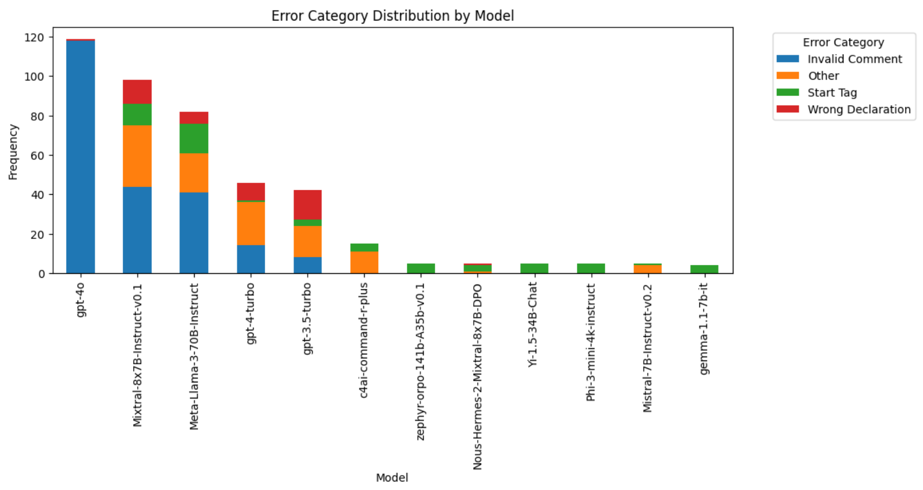


Fig. 5: Error Category Distribution by each of Checked LLMs.

Table 3: Success rate of the same task using different contexts, evaluated over N times.

Model	Context provided	Correct output	N
GPT-4o	Zero-shot	0	40
	Our one-shot prompt	2	40
	Our iterative prompting	32	40
meta-llama/Meta-Llama-3-70B-Instruct	Zero-shot	0	40
	Our on-shot prompt	21	40
	Our iterative prompting	30	40
mistralai/Mixtral-8x7B-Instruct-v0.1	Zero-shot	0	40
	Our one-shot Prompt	8	40
	Our iterative prompting	18	40

6 Conclusion

The rapid growth of LLMs enabled remarkable achievements for SE, including the ability to perform several code tasks based on text-only descriptions. For instance, it is possible to transform, add comments, think on and summarize code. In this paper, we tackled the DSLs' development following OMG MDA principles, *i.e.* starting with the design of meta-model for a future DSL. The result does respect a required output format and it is generated based on two input parameters: human friendly description and modelling language document type. However, the one-shot generation results often contain syntactic and semantics errors. We applied an iterative prompting approach and LLM agents to solve these issues. Detected errors are given back as input for the iterative process until correctness achievement. We checked our suggested process on several real-world case studies and performed an analytic study in order to show the advantages and actual LLMs limits to be dealt with in the future.

To sum up, while we believe that LLMs are potentially revolutionising software development, little attention in the literature is given to Model Based Software Engineering (MBSE). We also notice that there still have relevant questionable limitations to be addressed in term of insufficient training on specific languages, explicability, reliability, scalability, and resources consumption. Those challenges once dealt with, LLMs would be insightful assistant for model-based software engineering.

Aknowledgement: This work is supported and funded by IBCO-CIMI-CNRS research project (call 2021-2024).

References

1. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Extracting domain models from natural-language requirements: approach and industrial evaluation. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 250–260. ACM, Saint-malo France (Oct 2016). https://doi.org/10.1145/2976767.2976769, https://dl.acm.org/doi/10.1145/2976767.2976769
2. Combemale, B., Garoche, P.L., Crégut, X., Thirioux, X., Vernadat, F.: Towards a Formal Verification of Process Model's Properties - SimplePDL and TOCL Case Study. In: INSTICC (ed.) 9th International Conference on Enterprise Information Systems. pp. 80–89. INSTICC, Funchal, Madeira, Portugal (Jun 2007), https://hal.science/hal-00160807
3. Grose, T.J., Doney, G.C., Brodsky, S.A.: Mastering Xmi: Java Programming with Xmi, XML and UML, vol. 21. John Wiley & Sons (2002)
4. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H.: Large Language Models for Software Engineering: A Systematic Literature Review (Mar 2024), http://arxiv.org/abs/2308.10620, arXiv:2308.10620 [cs]

5. Jha, S., Jha, S.K., Lincoln, P., Bastian, N.D., Velasquez, A., Neema, S.: Dehallucinating Large Language Models Using Formal Methods Guided Iterative Prompting. In: 2023 IEEE International Conference on Assured Autonomy (ICAA). pp. 149–152 (Jun 2023). https://doi.org/10.1109/ICAA58325.2023.00029, https://ieeexplore.ieee.org/document/10207581
6. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 9459–9474. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf
7. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P.: Self-Refine: Iterative Refinement with Self-Feedback (May 2023). https://doi.org/10.48550/arXiv.2303.17651, http://arxiv.org/abs/2303.17651, arXiv:2303.17651 [cs]
8. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J.: Large language models: A survey (2024), https://arxiv.org/abs/2402.06196
9. Netz, L., Michael, J., Rumpe, B.: From Natural Language to Web Applications: Using Large Language Models for Model-Driven Software Engineering. pp. 179–195. Gesellschaft für Informatik e.V. (2024), https://dl.gi.de/handle/20.500.12116/43620
10. Patil, S.G., Zhang, T., Wang, X., Gonzalez, J.E.: Gorilla: Large Language Model Connected with Massive APIs (May 2023). https://doi.org/10.48550/arXiv.2305.15334, http://arxiv.org/abs/2305.15334, arXiv:2305.15334 [cs]
11. Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., Sun, M.: Communicative Agents for Software Development (Dec 2023), http://arxiv.org/abs/2307.07924, arXiv:2307.07924 [cs]
12. Saini, R.: Artificial intelligence empowered domain modelling bot. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. pp. 1–6. ACM, Virtual Event Canada (Oct 2020). https://doi.org/10.1145/3417990.3419486, https://dl.acm.org/doi/10.1145/3417990.3419486
13. Sun, R., Arik, S.O., Nakhost, H., Dai, H., Sinha, R., Yin, P., Pfister, T.: SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (Jun 2023), http://arxiv.org/abs/2306.00739, arXiv:2306.00739 [cs]
14. Wang, B., Wang, Z., Wang, X., Cao, Y., Saurous, R.A., Kim, Y.: Grammar Prompting for Domain-Specific Language Generation with Large Language Models (Nov 2023), http://arxiv.org/abs/2305.19234, arXiv:2305.19234 [cs]
15. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Jan 2023). https://doi.org/10.48550/arXiv.2201.11903, http://arxiv.org/abs/2201.11903, arXiv:2201.11903 [cs]
16. Yang, Z., Li, L., Wang, J., Lin, K., Azarnasab, E., Ahmed, F., Liu, Z., Liu, C., Zeng, M., Wang, L.: MM-REACT: Prompting ChatGPT for Multimodal Reasoning and Action (Mar 2023). https://doi.org/10.48550/arXiv.2303.11381, http://arxiv.org/abs/2303.11381, arXiv:2303.11381 [cs]