Journal on Software and Systems Modeling manuscript No.

On the Assessment of Generative AI in Modeling Tasks: An Experience Report with ChatGPT and UML

Javier Cámara · Javier Troya · Lola Burgueño · Antonio Vallecillo

Received: date / Accepted: date

Abstract Most experts agree that Large Language Models (LLMs), such as those used by Copilot and ChatGPT, are expected to revolutionize the way in which software is developed. Many papers are currently devoted to analyzing the potential advantages and limitations of these generative AI models for writing code. However, the analysis of the current state of LLMs with respect to software modeling has received little attention. In this paper, we investigate the current capabilities of ChatGPT to perform modeling tasks and to assist modelers, while also trying to identify its main shortcomings. Our findings show that, in contrast to code generation, the performance of the current version of ChatGPT for software modeling is limited, with various syntactic and semantic deficiencies, lack of consistency in responses, and scalability issues. We also outline our views on how we perceive the role that LLMs can play in the software modeling discipline in the short term, and how the modeling community can help to improve the current capabilities of ChatGPT and the coming LLMs for software modeling.

Keywords Large Language Models · ChatGPT · Software models · Modeling languages · UML

Javier Cámara
ITIS Software. Universidad de Málaga, Spain. jcamara@uma.es

Javier Troya
ITIS Software. Universidad de Málaga, Spain. jtroya@uma.es

Lola Burgueño
ITIS Software. Universidad de Málaga, Spain. lolaburgueno@uma.es

Antonio Vallecillo
ITIS Software. Universidad de Málaga, Spain. av@uma.es

1 Introduction

The emergence of Generative AI and Large Language Models (LLMs), such as those used by GitHub's Copilot [9] and OpenAI's ChatGPT [14], is causing quite a stir in the Computer Science community. Most experts foresee a major disruption in the way software is developed and software engineering education is also expected to drastically change with the advent of these LLMs [12]. These issues are a recurrent topic in many universities and are being covered by most specialized forums and blogs. A plethora of papers are now analyzing the potential advantages, limitations, and failures of these models for writing code [3], as well as how programmers interact with them [2,19]. Most studies seem to agree that LLMs do an excellent job in writing code: despite some minor syntactical errors, what they produce is essentially correct.

However, what about software modeling? What is the situation of LLMs when it comes to performing modeling tasks or assisting modelers to accomplish them? A few months ago we started looking at these issues, trying to investigate the current status of LLMs with respect to conceptual modeling, a topic that does not seem to have attracted much attention so far. Our premise is that LLMs are here to stay. So, instead of ignoring them or rejecting their use, we posit that it would be better to embrace and use them in an effective manner to help us perform modeling tasks.

We are aware that the current LLM situation is very volatile, with new models, versions and tools being released frequently, each one improving over the previous ones. However, our goal is to assess the current situation and to provide a set of experiments that can enable us to identify possible shortcomings of current tools for performing modeling tasks and assisting modelers, as

well as a way to measure the improvement of future versions.

In this paper, we focus on the development of software models and, more specifically, on how to build UML class diagrams enriched with OCL constraints. Of the existing LLMs, we will focus on ChatGPT, analyzing its possible use as a modeling assistant. To do so, we investigate several issues, such as: (1) the correctness of the UML and OCL models produced by ChatGPT; (2) the best way to ask ChatGPT to build correct and complete software models—in particular, UML class diagrams; (3) its coverage of different modeling concepts and mechanisms; (4) its expressiveness and cross-modeling language translation capabilities, and (5) its sensitivity to context and problem domains.

Our findings show that the performance of the current version[1] of ChatGPT's capabilities for software model development is not as good as for code generation. Our experiments concluded that ChatGPT is only able to deal with small models, and unable to properly handle some basic modeling concepts, such as association classes or multiple inheritance. The variability and inconsistency of the models produced in response to the same prompts was too high to ensure the repeatability and reproducibility of the results. Some obvious errors (such as associations that had composition symbols at both ends) were more frequent than expected. We also realized that the problem domain had a remarkable impact on the results. For example, in domains for which there is a large code base (e.g., banking), the models produced by ChatGPT had a very low level of abstraction, were very close to the programming level, and mostly correct. However, the models generated for more abstract domains, such as university courses or theater plays, were fundamentally flawed. In contrast, we found that ChatGPT's performance with OCL expressions and constraints was remarkable. We attribute this to the fact that OCL is very similar to SQL, for which there is an extensive base of programs on which ChatGPT seems to have been trained.

The structure of this paper is as follows. First, Section 2 introduces the context of our work and our main objectives. Section 3 describes the experiments we have conducted to understand the current capabilities of ChatGPT for performing modeling tasks. The results of these experiments are presented and analyzed in Section 4. Section 5 sets out our views of the present and foreseeable future of generative LLMs for performing software modeling tasks, how modelers can make the best use of them, and outlines some ideas on how the software modeling community can help to improve these tools. Finally, Section 6 concludes with some ending remarks.

[1] Stable release February, 2023.

2 Context

This section introduces the context of our work and our main objectives, formulated through a set of research questions.

2.1 AI-based assistant tools

Software assistants and conversational bots have been around for a long time [18]—think, for example, of Microsoft's infamous *Clippy*. However, they have not received much attention until recently, when their performance has been found to be outstanding and their responses have seriously challenged the Turing test in some instances. From the Arts to the Sciences, LLMs are demonstrating their great potential and value in helping with numerous tasks.

The way to use LLMs and interact with them depends on numerous factors. For example:

- *Interaction mode:* Interactions with assistants in software development are bimodal [2]: in *acceleration mode*, the programmer knows what to do next and uses a LLM such as Copilot or ChatGPT to get there faster; in *exploration mode*, the programmer is unsure about how to proceed and uses the assistant to explore options.
- *Type of assistance:* We can distinguish between two types of AI-based tools for software modeling depending on their use. First, there are auto-completion wizards that propose new classes, attributes and relationships while the model is being developed, e.g., [4, 6, 7, 16, 17, 20]. Second, there are tools that can be asked to perform the complete task, and then the user can sometimes refine or extend the tool's results based on their correctness, completeness or suitability, if needed. Examples of such tools are Copilot and ChatGPT.

LLMs are deep learning models trained with massive datasets to perform specific tasks. They all incorporate from millions to billions of parameters that, in some occasions, can be fine-tuned to be adapted to problems similar to those for which they have been initially trained. Usually, these models contain a series of hyperparameters that allow users to customize the predictions. The choice of good hyperparameter values has an important impact on the quality of the results. An appropriate hyper-parametrization for a specific task could be as important as the dataset used for training—see, e.g., [8] on how the hyper-parametrization of LLMs such as Copilot or Codex affects their results. However, it is not clear whether the advantages of choosing the most appropriate hyperparameters for the task at hand outweigh

their limitations, in terms of the needed knowledge and skills, complexity, required effort, and payoff in the results. For instance, tools such as ChatGPT do not allow users to configure their hyperparameters and these are inferred from the prompt. In contrast, the new Bing search engine[2] allows the non-expert user to set only a few hyperparameters, but not all. For this, Bing has modified how the hyper-parametrization of the LLM is done and allows the user to choose the conversation style with three options: "more creative", "more balanced" and "more precise", instead of asking them to select a value (i.e., the so-called `temperature` value) within a given interval, usually a Real number between 0 and 1.

2.2 ChatGPT

ChatGPT is a tool developed by OpenAI, a for-profit research organization co-founded by Elon Musk and Sam Altman, strongly funded by Microsoft. The users interact with ChatGPT in a conversational way via text prompts.

When asked about its modeling knowledge, ChatGPT reports that it knows most UML diagrams, including Class diagrams, Use cases, State machines, Sequence diagrams, and Activity diagrams.

Regarding the UML notations ChatGPT can handle, being a language model, it cannot generate models in graphical form. ChatGPT produces models in textual UML notations, including PlantUML, USE (the UML-based Specification Environment), Yuml, Markdown UML, Mermaid and UMLet. It also produces some rudimentary class diagrams using plain characters to draw boxes and lines, but sometimes these are difficult to parse and understand. Fig. 1 shows one example of these textual diagrams.

We discovered that ChatGPT can also handle Ecore models. You can ask it to generate models in Ecore and also use them as inputs for prompts. Its treatment of the Ecore language is comparable to that of other modeling languages, with similar mistakes and correct answers.

We also asked ChatGPT about other textual languages that it knows, which are used in UML for representing different aspects of software systems. It mentioned the Object Constraint Language (OCL), the Action Language for Foundational UML (ALF), the UML Profile Definition Language (UML PDL), and the UML Testing Profile (UTP). We checked in depth its skills with OCL, which are excellent, but in contrast, the initial tests with the other notations did not yield satisfactory results.

[2] `https://www.bing.com/search?q=Bing+AI&showconv=1&FORM=hpcodx`

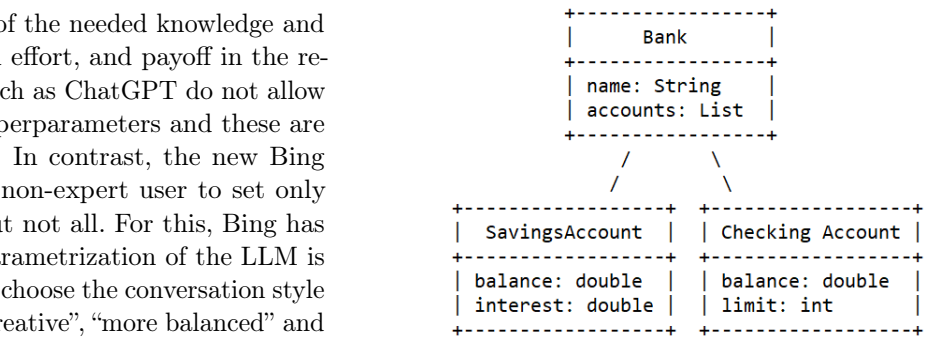

Fig. 1 A textual diagram generated by ChatGPT.

2.3 Research questions

As mentioned in the introduction, our primary goal was to analyze the use of ChatGPT as an assistant tool for conceptual modeling. In line with this, we address the following Research Questions:

RQ1. *Does ChatGPT generate syntactically correct UML models?*
RQ2. *Does ChatGPT generate semantically correct models, i.e., semantically aligned with the user intents?*
RQ3. *How sensitive is ChatGPT to the context and to the problem domain?*
RQ4. *How large are the models that ChatGPT is able to generate or handle?*
RQ5. *Which modeling concepts and mechanisms is ChatGPT able to effectively use?*
RQ6. *Does prompt variability impact the correctness/quality of the generated models?*
RQ7. *Do different use strategies (e.g., prompt partitioning) result in different outcomes?*
RQ8. *How sensitive is ChatGPT to the UML notation used to represent the output models?*

To answer these research questions, we devised a set of experiments, which are detailed in the next section.

3 Experiments

This section describes the experiments we conducted to understand the current capabilities of ChatGPT to perform modeling tasks. We defined two phases. In the first one, we carried out some exploratory experiments to gain a basic understanding of how ChatGPT works with software models, as well as its main features and limitations. The experiments in the second phase were more systematic and aimed to further characterize ChatGPT's modeling capabilities. The results of these experiments are presented and discussed later in Section 4.

3.1 First phase: exploration

Objective. In this *exploratory* phase, the four authors of this paper interacted individually with ChatGPT to become acquainted with its modeling capabilities. We also explored some of its general characteristics. Since we are not able to set hyperparameters such as the number of tokens, we explored the size of the models it was able to handle. We also explored its skills with various modeling notations, which depend on the training data.
Method. For this phase, we did not use any systematic approach but tried to explore all the ideas that came to mind based on the findings we were making and the results we were obtaining.
Materials. We wrote prompts asking ChatGPT to create models of different sizes, as well as to create the target models of some of the assignments that we use in our modeling lectures. The size of these models ranged from 10 to 40 classes and associations. We wrote all our interactions and findings in a shared document used as a logbook [1].
First findings. We became aware of several basic capabilities and limitations of ChatGPT. Some of them were not surprising, given how language models work, but they are still worth reporting here.

F1. Problem domain and semantics: The problem domain is important for ChatGPT. In general, it works poorly when the names of the entities to be modeled have no meaning, such as X, Y, Z, or A, B, C. The more meaningful and representative entity names are, the better the class model it produces. Similarly, the more ChatGPT "knows" about the domain, the more accurate and complete the UML model it generates. Purchase Orders, Banks, or Employees are concepts for which it is able to produce semantically rich models (too rich sometimes, as it completes them with information that was not requested).

F2. Problem domain and syntax: The problem domain also seems to influence the structure and contents of the resulting models, as well as their level of abstraction. In some domains, the models generated had a very low level of abstraction, quite close to a software program represented in UML. In others, the level of abstraction was higher, although it heavily depended on the particular conversation. As we know, LLMs have semantic and syntactic capabilities. When mixing these two abilities to produce class models, depending on the concrete domain (and thus the amount of data about that domain in the training dataset), ChatGPT seems to rely on its translation capabilities. Sometimes, given our prompt, ChatGPT's outputs seem to be the UML representation of a possible solution that it found/produced in a different language, i.e., with a different syntax. If this other language is a low-level language such as Java or C++, the abstraction level is lower than if it finds a solution represented as a software model such as a relational schema. In other words, the problem domain influences the result, as the latter depends on the data with which ChatGPT has been trained for that domain.

F3. Publicly available models: Related to the previous point, if you ask ChatGPT to build a UML model that is on the Internet (such as the example given in the OCL 2.4 standard), ChatGPT will generate a correct model. OpenAI has not disclosed what data was used to train ChatGPT or how the training process was conducted, but it looks like these publicly available models have served as training models for ChatGPT.

F4. Size of the models to build: The current version of ChatGPT does not work well when asked to generate a class model of more than 8-10 classes from scratch. However, it works much better if you ask it to build a small initial model and progressively add information to it. In fact, ChatGPT was unable to cope with any of the exams of our modeling course, because these UML models were too large (more than 20 classes and associations) for its current capabilities or hyperparametrization, and it either did not finish the task (which had to be aborted) or built rather small and incomplete models.

F5. Notations: We also experimented with various notations to represent the generated UML model. By default, ChatGPT seems to use a diagrammatic notation that employs characters to draw boxes and lines on the screen. This notation is too difficult to read and understand when there are more than four or five classes in the model, so we started to explicitly ask ChatGPT to produce models in specific notations, such as PlantUML or USE. Apart from small syntactic errors, the results are generally good; we cannot say the same for the semantics of the generated models, which were full of errors, as we shall later see.

F6. Conversation history: Although there is a limit to the amount of information ChatGPT can retain, it is able to "remember" what was said earlier in a conversation. This is, ChatGPT is conversation-aware and results are heavily conversation-dependent.[3] Depending on the session, and on our previous interactions, the results may present remarkable variations. In fact, when asked to build a model, ChatGPT takes information from previously developed models within the same conversation, even if they have nothing to do with the model in question. This is why it is important to start a new chat every time we want to develop a new model. One exercise we did was to ask ChatGPT to generate a UML model in three different chats using the same prompt.

[3] OpenAI states that, when replying to a prompt, ChatGPT does not access previous conversations.

In two of them we had been previously creating models from other domains, and the third chat restarted afresh. The results generated in the first two conversations were very similar to the previously generated models, despite the fact that the new model was from a different domain. The results of the same prompt in the new chat were closer to the desired target.

F7. Cross-language translation facilities: When testing the translation facilities across modeling languages, the results are conversation-dependent. For example, we gave ChatGPT a model in USE with association classes, and asked it to represent the model in PlantUML. The result was not correct, because ChatGPT does not seem to know how to handle association classes. Now, given that same PlantUML model, if asked to convert it to USE, depending on whether it is within the same conversation or in a different one, sometimes ChatGPT converts it to the original USE model (even with association classes) or to a different model (this time with syntactic errors in USE). Interestingly, this does not seem to be specific only to modeling, but also to translation between other languages, even natural ones.

F8. Integrity constraints: When the description of the model to be represented includes integrity constraints (which we would expect to be specified by means of OCL expressions), what ChatGPT usually does for each constraint is either to create a note or to define an operation that checks the constraint on the class that would correspond to the context of the OCL expression. We soon learned that if what we want to represent are the integrity constraints of a UML class model using OCL, it is better to develop the model without constraints, and then explicitly ask ChatGPT to generate the constraints in OCL, one by one. ChatGPT works significantly better with OCL than with UML. We suspect that this is possibly due to the fact that the data sources used for the construction of OCL expressions are usually SQL, Rust, and other declarative languages for which there is a much larger corpus than for UML.

3.2 Second phase: focused experiments

Objective. In the first phase we managed to obtain a basic understanding of how ChatGPT works, as well as of its main features and limitations. We also obtained initial responses to some of the research questions, namely those about its sensitivity to context and problem domain (RQ3, addressed by findings F1, F2, F3 and F6), its scalability (RQ4, addressed by finding F4), and partly about its sensitivity to the modeling notation of choice (RQ8, addressed by findings F5, F7 and F8). The goal of this second phase was to address the rest of the research questions, which demanded a more systematic approach.

Method. For this phase, we developed a set of models that were intended to cover the most important modeling concepts and mechanisms (see left column of Table 1). Each author independently proposed ten UML models. All of them were small in size (three to six classes) so that ChatGPT could handle them without problems. They represented different *user intents*, and for each one of them, the exercise consisted in asking ChatGPT to produce the corresponding UML model using one or more prompts.

Figure 2 shows one of these exercises (Video club). The prompt used to generate the UML class diagram is shown on the left and the ChatGPT response (in PlantUML) is shown on the right. For readability purposes, we have included the graphical representation of the PlantUML model in the central box. On this occasion, ChatGPT managed to generate the intended model after a few interactions, so the exercise was considered successful. However, to illustrate the variability of ChatGPT's responses, Figure 3 shows another model generated by ChatGPT in response to exactly the same prompt, but from a different conversation (both were fresh conversations). Although there are deterministic language models, most modern LLMs (such as ChatGPT) are designed to be probabilistic, and not deterministic. This lack of repeatability of the results represents a major obstacle to the reproducibility of the experiments, and is one of the main current challenges of these assistants from our point of view.

Even if prompts were carefully designed, very often ChatGPT did not generate the expected result. To improve the result, we always tried to follow a conversation with the bot by providing multiple successive prompts in which we asked to modify some aspect of the generated result. For example, if ChatGPT generates a class `Movie` that does not contain an attribute `name`, we can tell ChatGPT that movies must have a name. The same can be done to add the multiplicities and role names of the associations, remove unwanted methods, or fix incorrect details (such as using compositions when there is a multiplicity `1..*` in the composite end As we will mention later, ChatGPT does not always fix or add what we ask for such as repairing the multiplicity of an association. When it does it, ChatGPT sometimes introduces additional errors in other parts of the model.

From the complete set of 40 exercises, we selected two from each author. The resulting eight models covered the concepts and mechanisms listed in Table 1. Their intent models are shown in Figure 4.

Each author tried to make ChatGPT generate these UML intent models as faithfully as possible, using different strategies to create the prompts. A summary of the results of this experiment is shown in Table 2. The

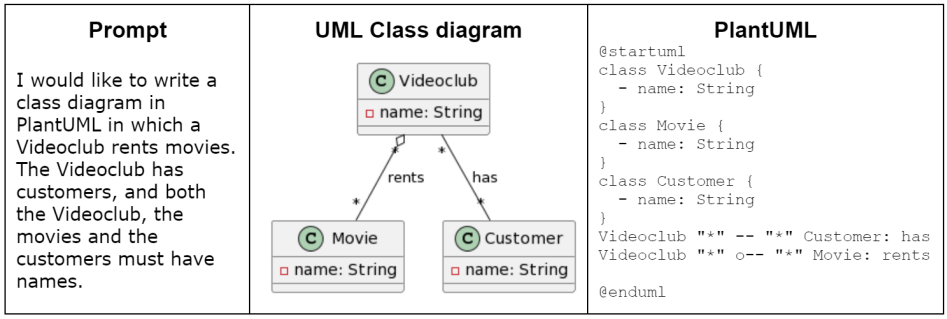

Fig. 2 Prompt used to ask ChatGPT to generate a UML class diagram of a video club system, and the resulting model.

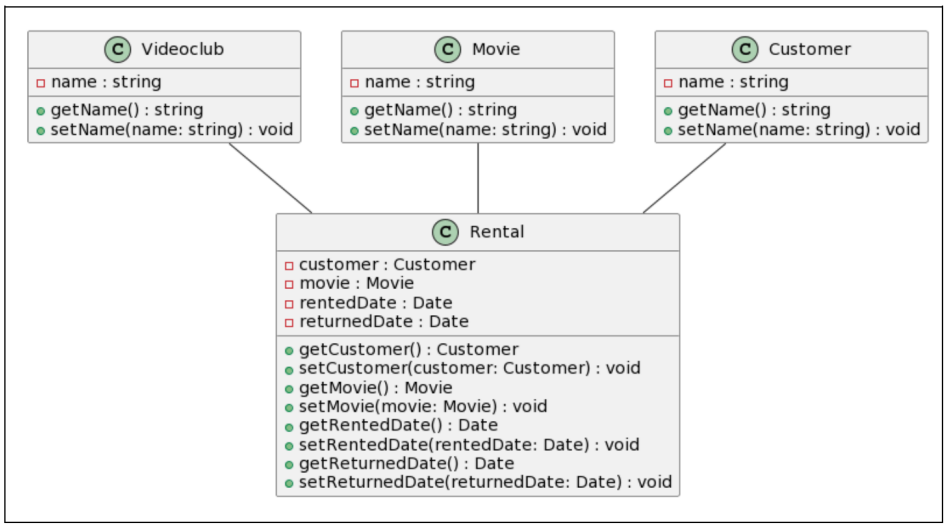

Fig. 3 Another model generated by ChatGPT in response to exactly the same prompt, but in a different session.

Table 1 Coverage by the selected examples of the main modeling concepts and mechanisms.

Concept/Mechanism	Students	Airlines	File System	Robots	Video club	Theaters	Amphibious	Cars
Enumerations				X		X	X	
Classes	X	X	X	X	X	X	X	X
Attributes	X	X	X	X	X	X	X	X
Operations		X						
Generalization			X	X		X	X	
Association	X	X	X	X	X	X		X
Aggregation	X		X	X		X		
Composition								X
Assoc. Class		X		X		X		
Multiple inheritance							X	
Abstract classes				X		X	X	
OCL constraints							X	X
Roles (as assoc. ends)								
Roles (as inherited classes)						X		
Roles (as entity types) [5]								
Materialization [15]						X		X

columns list the exercise, the number of authors that could make ChatGPT successfully generate the intended model, the average number of sessions that were used, and the average number of prompts that were required per session until the solution was generated or the author gave up. Reasons for restarting a new chat or giving up included that: (1) ChatGPT entered an endless loop, e.g., saying "Sure, I will fix it" but repeating the previous response, and (2) class diagrams that had an increasing number of errors despite our indications to fix them, or diagrams that were not worth fixing.

Materials. The complete set of UML models of the 40 exercises is available from our GitHub repository [1], as well as the reports that each author produced during their interactions with ChatGPT.

Findings. The exercises of this phase revealed some very interesting findings, which are summarized below.

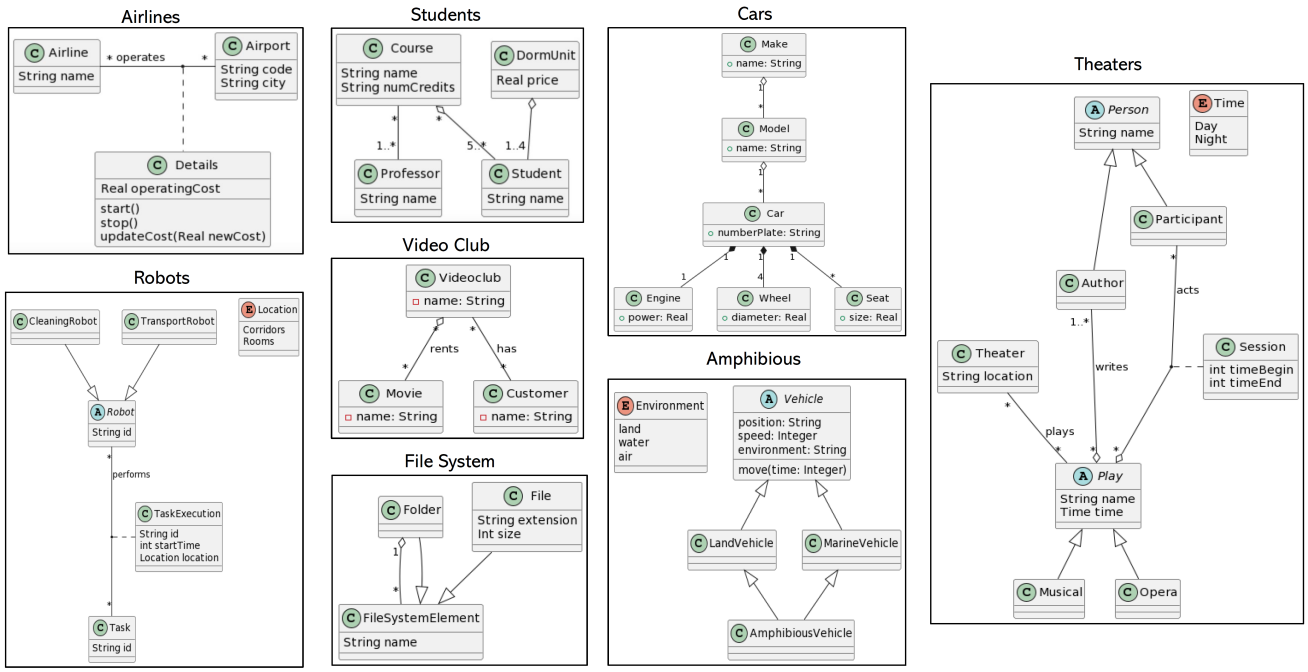

Fig. 4 Intent models of the eight selected exercises.

Table 2 Results of the experiment where the four authors tried to make ChatGPT generate the intent models of the selected exercises.

Exercise	Successful	Avg. Sessions	Prompts/Sess.
Students	4/4	2.5	2.5
Airlines	0/4	3	2.75
File System	4/4	2	2.25
Robots	0/4	3	3.5
Video club	4/4	2	2.3
Theaters	0/4	3	3
Amphibious	4/4	2.2	1.75
Car Parts	4/4	2	2.3

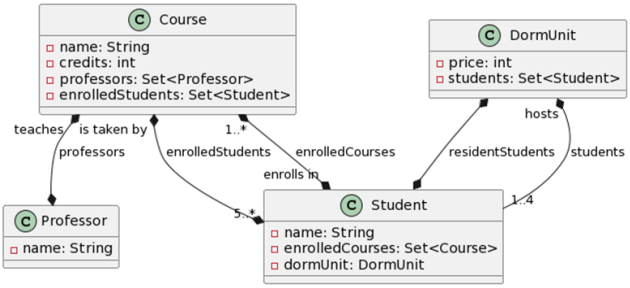

Fig. 5 Example showing some of the mistakes made by ChatGPT when representing associations (real ChatGPT output).

F9. Relationships: ChatGPT is able to capture associations and inheritance adequately, although not always. The ability seems to depend on the domain being modeled. Modeling of role names, on the other hand, seems to work well for most domains.

F10. Determinism: Results are rather random, with major differences for the same prompt in different chats, as discussed above and illustrated in Figures 2 and 3.

F11. Semantics: Syntactically, results are mostly correct. However, semantically they are not always correct. Examples of common mistakes include:

1. Duplicating aggregations (and sometimes even associations) by defining, in addition to the association, an attribute in the containing class with the list of related elements, which is equivalent to the association and therefore redundant.
2. Mistakenly modeling relations as directed associations. When asked to convert them into bidirectional associations, two opposing directed associations are created. These cannot be merged later, even if we explicitly ask ChatGPT to do so.
3. Creating compositions or aggregations with two composite ends, as illustrated in the example shown in Fig. 5. Note that the multiplicities of these relations are semantically incorrect, too.

F12. Iterative process is required: Several iterations with explicit requests for modification are usually needed to approximate the user intent model (cf. Table 2). Thus, the task of developing a model usually consists of a dialogue with ChatGTP, rather than a single request-response interaction. Normally, we start with an initial prompt and refine the result until we achieve the desired intent model. Given the large variability of ChatGPT's responses to exactly the same prompt, it is even a good strategy to start several conversations and continue with the one whose initial model is most promising, both re-

garding its level of abstraction and its contents (classes, attributes and associations). This is also important because the iterative process does not always converge. Sometimes the requested changes were implemented so poorly that the models became badly flawed and we had to start from scratch (or from some regenerated model in an intermediate step of the dialogue).

F13. Constructs of the UML language not handled properly: There are UML constructs that ChatGPT can incorporate but does not always handle adequately. For example, if we give it a UML model with an association class, ChatGPT is able to handle it and even define correct OCL constraints involving that association class. However, none of the models that ChatGPT generates include association classes, even when they would be the most natural way to model the problem. We also tried to give ChatGPT a USE model that contained an association class and asked it to write the model in PlantUML. ChatGPT converted it to a model without association class. We explicitly asked ChatGPT to rewrite the model to have association classes, and it said yes, but did not. In fact, none of the three intent models of the experiment containing association classes could be created with ChatGPT: Airlines, Robots and Theaters (cf., Table 2).

F14. Enumerations: In most cases, enumerations are not used by ChatGPT unless explicitly requested. It rather uses either inheritance or strings. Unlike with association classes, when explicitly asked to use enumerations, it does so correctly.

F15. Multiple inheritance: Multiple inheritance is not handled correctly. We needed to explicitly describe the type of relationship and what the source and target classes were to obtain the desired result. Although ChatGPT most times ends up producing the right result, there is high variability in its responses, producing correct and incorrect models seemingly at random.

F16. OCL constraints: Initially, ChatGTP does not include constraints in the model even when they were stated in the prompt. When explicitly asked to include them, ChatGPT first proposes using notes, and then operations. When we asked ChatGPT about whether OCL could be used instead, mostly correct OCL constraints were generated (apart from minor syntactic mistakes on a few occasions).

F17. Capacity for abstraction: ChatGTP (unlike human modelers) has no capacity for abstraction. If it is asked to represent the UML model of a car with four wheels, it sometimes creates four such attributes, as opposed to a more general form of modeling that is capable of using a collection of wheels that now has four but at another time might have more or less. For a small number of elements, this strategy is acceptable, but it is suboptimal when the number increases above a certain threshold. Similarly, ChatGPT does not factor out the common attributes of subclasses and place them in the superclass on its own.

F18. Effort required by the modeler: Finally, the amount of time and effort required to produce the correct intent models is not negligible, especially considering the small size of these models. For example, in all the intent models that could be correctly produced, the total number of interactions with ChatGPT (counting the prompts of all sessions until the model was correct) exceeded the number of model elements.

4 Analysis

After carrying out the experiments and analyzing our experience with ChatGPT, this section is dedicated to answering the research questions identified in Section 2.3.

RQ1. *Does ChatGPT generate syntactically correct UML models?*

The UML models produced by ChatGPT are generally correct, although they may contain small syntactic errors (see finding F5). This also depends on the notation used. Although we did not test it thoroughly, the level of syntactic correctness of the models produced in PlantUML was much higher than those generated in USE, for example.

RQ2. *Does ChatGPT generate semantically correct models, i.e., semantically aligned with the user's intent?*

This is the weakest point that we observed during our interaction with ChatGPT. Some studies suggest that LLMs are better at syntax than producing semantically correct results [11]. Our findings (e.g., F13) corroborate this fact. This includes errors in both the semantics of the language and the semantics of the domain being modeled. On many occasions we observed that ChatGPT proposed seemingly random models that made no sense from either a modeling or domain standpoint.

RQ3. *How sensitive is ChatGPT to the context and to the problem domain?*

Our findings F1, F2, F3 and F6 clearly show that not only the problem domain influences the resulting models, but also the information exchanged during the dialogues with ChatGPT. In addition, the more ChatGPT "knows" about a domain (i.e., the more data about a domain was used during training), the closer-to-correct class models it produces. ChatGPT produces its worst results when

it has little or no information about the domain or the entities to be modeled, as it happened when asked to produce software models of entities such as *Snarks* or *Zumbats*, for which it did not seem to have any reference or semantic anchor.

RQ4. *How large are the models that ChatGPT is able to generate or handle?*

As mentioned in Finding F4, ChatGPT currently has strict limitations on the size of the models it can handle. It has serious problems with models larger than 10-12 classes. Even the time and effort required to produce smaller models (Finding F19) are not insignificant.

RQ5. *Which modeling concepts and mechanisms is ChatGPT able to effectively use?*

The modeling concepts that we analyzed were shown in Table 1. There is a high degree of variability in how ChatGPT handles them. We observed that it is able to manage reasonably well (with some exceptions) associations, aggregations and compositions, simple inheritance, and role names of association ends (F9). However, it requires explicit indications for using enumerations (F14), multiple inheritance (F15) and integrity constraints (F16). Finally, we found out that its results are not acceptable when using abstraction (F17), and it cannot handle association classes (F13).

RQ6. *Does prompt variability impact the correctness/quality of the generated models?*

We observed that there is plenty of variability when ChatGPT generates responses to same prompt (F10). We learned that it is useful to start a new conversation from scratch when the results were not good, in order to find better solutions for the same intent model (F12).

RQ7. *Do different use strategies (e.g., prompt partitioning) result in different outcomes?*

First, as noted in finding F4, the size of the models that ChatGPT is capable of handling in a single query forces the modeling task to become an iterative process in which the user starts with a small model and progressively adds details to it (F12). The variability and randomness of ChatGPT responses (F10) or when results within a conversation start to diverge often force the modeler to repeat conversations to try to obtain better models.

RQ8. *How sensitive is ChatGPT to the UML notation used to represent the output models?*

ChatGPT is capable of representing models with several notations (F5), although in general it makes fewer syntactic mistakes with PlantUML. It is also much better with OCL than with UML (F8). Finally, we also looked at how accurate ChatGPT was with cross-modeling language translation (F7), realizing that this task works better within the same conversation, but not across conversations.

5 Discussion

From our study, we conclude that ChatGPT is not yet a reliable tool to perform modeling tasks. Does that mean we should discard it, or at least wait to see how it evolves before taking any action? Our position is that, on the contrary, we should start working now to improve the modeling skills of ChatGPT and other LLMs to come, and to build a future where these assistants are destined to play a prominent role in modeling.

This section sets out our views about the future of LLMs that we foresee when it comes to performing software modeling tasks, and about how modelers can make the best use of them. It is divided into three parts. First, we describe the Model-Based Software Engineering (MBSE) tasks in which LLMs can be helpful, and how we can use LLMs to accomplish them. Second, we discuss the consequences that the new *status quo* may have on the way we develop models and teach modeling, including the new possibilities it opens and the new roles that software engineers could play in this new context. Finally, we discuss what we think is needed to realize this vision.

5.1 The role of assistants in MBSE

In our opinion, ChatGPT or any other LLM can be of invaluable help in many areas of MBSE, complementing the current work of software modelers and letting them focus on the tasks for which they really provide value.

Model development. LLMs can help develop models both in acceleration and exploration modes [2]. Modelers typically generate models by composing (usually in their heads) model *fragments*, each of which addresses a concern or implements a feature. These model fragments are reused from existing conceptual patterns or solutions known to the modeler, adapting them to the problem at hand. Assistants could be of great help in this case, identifying these existing patterns or solutions and automatically performing the adaptation. For example, in acceleration mode, the tool can provide solutions to add security aspects to a model, extend an existing model to implement more entities or functionalities, or provide model elements with new features, among other tasks. In exploration mode, an LLM can provide a set

of options to a modeler on how to model certain system aspects. For example, whether it would be better to use association ends, inheritance or entity types to model certain roles in the application. We could also ask the LLM about how to model certain requirements, and ask it to add to our model the one that best suits our needs. In this context, the modeler would identify the features or functionalities to be incorporated into the model, using natural language, and the wizard would be in charge of automatically adding them, until the model is complete.

Another task in which LLMs could be very useful is the generation of object models that conform to a given class diagram. We have tested this functionality in ChatGPT and the results have been very good, although we found similar problems to those we had during model generation. Namely, ChatGPT is able to produce very diverse instance models quickly and efficiently, although the quality of those models is not optimal. For example, most of them do not respect the integrity constraints of the class diagram. As soon as the quality of ChatGPT improves, e.g., by including some grammatical checks such as those available for SQL [10], it could outperform the current instance model generators, thus successfully taking on this tedious and costly task.

Model-based testing. In addition to generating sets of instance models from a UML class model, that could serve as test inputs, LLMs can also be used to generate test cases for the system. For several simple systems (such as a bank account, a microwave, an online shopping system, and a flight reservation system) we gave ChatGPT class diagrams with the specification of their structure and operations, and state machines with the specification of their behaviors, and asked it to generate test cases for them. The results were very accurate and complete, covering all relevant cases. Investigating in depth the quality of the test cases that ChatGPT is able to generate is part of our future work.

MBSE Education. The methods for teaching modeling are likely to be one of the things that will change the most. A few ways in which LLMs can be used to improve modeling education include:

- *Enhanced Learning*: LLMs can help students to learn modeling languages by providing real-time feedback on syntax, highlighting common errors, and offering suggestions for improvement. Additionally, they can provide contextual help, e.g., providing definitions and examples of modeling concepts.
- *Model Completion*: LLMs can provide auto-complete functionality when students are developing models, which can save time and improve accuracy.
- *Model Generation*: LLMs can also generate models based on natural language descriptions. This can be useful for students who are just starting and may not yet be familiar with modeling, or with the syntax of a particular modeling language.

In addition, other tasks where LLMs could be of great help—although they would require more elaborate tool support—are the following.

- *Personalized Learning*: As with other subjects, LLMs can be used to provide personalized learning in computer science education. If complemented by a tool that analyzes the student's strengths, weaknesses and learning style, LLMs can provide tailored instruction and feedback that meets the individual needs of students.
- *Automated grading and assessment*: LLMs can provide instant feedback to students on their performance. This can save teachers time and help them provide more effective feedback to students.

5.2 How will the game change?

Overall, the use of large language models has the potential to revolutionize software modeling engineering and education, making it more accessible, personalized, and efficient. To get to that point, we will first need to improve the current consistency and reliability of the models produced by LLMs such as ChatGPT. Second, we will need to change the way in which we currently develop software models and teach modeling. These two issues are described in the following.

First of all, modeling assistants will become key components in model development processes. Software modelers will be able to interact with them in natural language in order to build and test their models. For example, modelers may rely on LLMs to explore modeling choices, add new features to a model, or change a model to accommodate to new or evolving requirements.

Secondly, new software engineering roles will also appear. For example, companies have started incorporating the new role of *prompt engineer* [13], whose job is to test AI chatbots using natural language instead of code. Their goal is to identify both errors and hidden capabilities so that developers can either fix or exploit them. They are also experts on how best to ask an LLM to perform a particular task so that it is carried out in the most accurate and efficient manner by the chatbot.

New opportunities also emerge for experts in *configuring the hyperparameters* that allow users to customize the LLM predictions in order to improve the quality of the results. As mentioned earlier, an appropriate hyperparametrization for a specific task could be as important

as the dataset used for training the LLM [8] or the actual choice of the (deep learning) algorithm. Similarly, *LLM trainers* can help provide the appropriate datasets to improve the prediction accuracy of an LLM in particular domains, and for specific tasks.

MBSE educators will have to change the way they perform most of their tasks today. Since LLMs will be ubiquitous, professors will not be able to prevent students from using LLMs for their assignments. On the contrary, one of their goals will be to help students use modeling assistants in the best possible way to learn new concepts, develop software models, and test them. In addition, they will need to help students to develop critical thinking skills that enable them to distinguish when the information provided by an assistant is useful and correct and when it is not.

Finally, researchers and academics will be able to use LLMs to analyze large amounts of models, identify patterns and insights, and generate new ideas from them.

5.3 How do we make this happen?

The prospects are certainly encouraging. The question is whether they are really attainable and, if so, how they can be achieved. It is clear that ChatGPT's abilities to perform modeling tasks are not yet up to the job. In this section, we would like to propose some suggestions that the modeling community could implement to improve the reliability and accuracy of ChatGPT and other Generative AI models.

First, we should make more (correct) software models available in public repositories, thus increasing the accessibility of datasets that can be used for training LLMs and other Generative AI models. The more UML and software models that are publicly available from different domains, the more accurate and reliable the responses from these AI models will be.

Second, we should start using LLMs/Generative AI models in our software modeling tasks to familiarize ourselves with them, explore their possibilities and discover their limitations. We should strive to use them not only for developing software models, but also for testing them, generating instances and test cases, etc. Exploring their use for other MBSE tasks and activities could also be valuable. We are sure that AI models can open new ways to make use of models in software and systems engineering tasks.

Providing feedback to the results of AI models, whenever available, will benefit the whole community. Training them should become a community effort, i.e., a responsibility of each and every one of us.

Developing a body of knowledge that incorporates a set of guidelines about the best strategies to interact with AI-based assistants for various types of modeling tasks, as well as a catalog of capabilities and common limitations, can also contribute to streamline the assimilation of AI models for modeling tasks.

Finally, let us incorporate LLMs and Generative AI models into our teaching practices. Making students acquainted with them and aware of their possibilities and limitations will help them not only to improve their modeling skills, but also their critical thinking. They should learn to discriminate when to use these AI models and when not to, as well as when to trust their answers.

6 Conclusions

Generative AI and Large Language Models are becoming ubiquitous, and their upcoming impact on our disciplines and professions cannot be overlooked. In this paper, we have investigated their current capabilities and limitations for generating UML class diagrams and for assisting software engineers to perform modeling tasks. Our findings show that, in contrast to code generation and completion, the performance of the current version of ChatGPT for software modeling is still quite limited.

Our intention was not to conduct an exhaustive set of experiments regarding the capabilities of LLMs for assisting in modeling tasks, as they are currently changing very fast. However, we wanted to address the growing need to have a picture of their current state, as accurate as possible. We also did not want to address other issues related to these types of tools, such as their ethical concerns. Although equally important, in this article we have focused mainly on their technical aspects.

In general we believe that, far from detracting from the use of this type of generative AI-based tools, we should try to help improving them as much as possible. In addition, we should start adapting our model-based engineering practices to these new assistants and the possibilities they offer. Likewise, we should start changing our modeling education methods to incorporate them.

Successfully addressing the challenge of seamlessly integrating these new LLMs and Generative AI models into our MBSE methods and practices is crucial. It could significantly increase the impact of MBSE on society and lead to a major step forward for our profession.

Acknowledgements We would like to thank Jörg Kienzle for his comments and very valuable feedback on an earlier draft of this paper. This work was partially funded by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación–Agencia Estatal de Investigación) under projects PID2021-125527NB-I00 and TED2021-130523B-I00.

References

1. Atenea Research Group: Git repository: chatgpt-uml (2023). URL https://github.com/atenearesearchgroup/chatgpt-uml
2. Barke, S., James, M.B., Polikarpova, N.: Grounded copilot: How programmers interact with code-generating models. CoRR **abs/2206.15000** (2022). URL https://arxiv.org/abs/2206.15000
3. Borji, A.: A categorical archive of chatgpt failures. CoRR **abs/2302.03494** (2023). URL https://arxiv.org/abs/2302.03494
4. Burgueño, L., Clarisó, R., Gérard, S., Li, S., Cabot, J.: An NLP-based architecture for the autocompletion of partial domain models. In: Proc. of CAiSE'21, *LNCS*, vol. 12751, pp. 91–106. Springer (2021). DOI 10.1007/978-3-030-79382-1_6
5. Cabot, J., Raventós, R.: Roles as entity types: A conceptual modelling pattern. In: Proc. of ER'04, *LNCS*, vol. 3288, pp. 69–82. Springer (2004). DOI 10.1007/978-3-540-30464-7_7
6. Capuano, T., Sahraoui, H.A., Frénay, B., Vanderose, B.: Learning from code repositories to recommend model classes. Journal of Object Technology **21**(3), 3:1–11 (2022). DOI 10.5381/jot.2022.21.3.a4
7. Chaaben, M.B., Burgueño, L., Sahraoui, H.: Towards using few-shot prompt learning for automating model completion. In: Proc. of ICSE (NIER)'23. IEEE/ACM (2023)
8. Döderlein, J., Acher, M., Khelladi, D.E., Combemale, B.: Piloting copilot and codex: Hot temperature, cold prompts, or black magic? CoRR **abs/2210.14699** (2022). URL https://arxiv.org/abs/2210.14699
9. GitHub: Copilot: Your AI pair programmer (2023). URL https://github.com/features/copilot/
10. Kim, H., So, B.H., Han, W.S., Lee, H.: Natural language to SQL: Where are we today? Proc. VLDB Endow. **13**(10), 1737–1750 (2020). DOI 10.14778/3401960.3401970. URL https://doi.org/10.14778/3401960.3401970
11. Marcusarchive, G., Davisarchive, E.: GPT-3, Bloviator: OpenAI's language generator has no idea what it's talking about (2020). URL https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/
12. Meyer, B.: What Do ChatGPT and AI-based Automatic Program Generation Mean for the Future of Software. Commun. ACM **65**(12), 5 (2022). URL https://cacm.acm.org/blogs/blog-cacm/268103-what-do-chatgpt-and-ai-based-automatic-program-generation-mean-for-the-future-of-software/fulltext
13. Mok, A.: 'Prompt engineering' is one of the hottest jobs in generative AI. Here's how it works. Business Insider (2023). URL https://www.businessinsider.com/prompt-engineering-ai-chatgpt-jobs-explained-2023-3
14. Open AI: ChatGPT (2023). URL https://chat.openai.com/chat
15. Pirotte, A., Zimányi, E., Massart, D., Yakusheva, T.: Materialization: A powerful and ubiquitous abstraction pattern. In: Proc. of VLDB'94, pp. 630–641. Morgan Kaufmann (1994). URL http://www.vldb.org/conf/1994/P630.PDF
16. Rocco, J.D., Sipio, C.D., Ruscio, D.D., Nguyen, P.T.: A GNN-based recommender system to assist the specification of metamodels and models. In: Proc. of MODELS'22, pp. 70–81. IEEE (2021). DOI 10.1109/MODELS50736.2021.00016
17. Saini, R., Mussbacher, G., Guo, J.L.C., Kienzle, J.: Automated, interactive, and traceable domain modeling empowered by artificial intelligence. Software and Systems Modeling **21**(3), 1015–1045 (2022). DOI 10.1007/s10270-021-00942-6
18. Savary-Leblanc, M., Burgueño, L., Cabot, J., Pallec, X.L., Gérard, S.: Software assistants in software engineering: A systematic mapping study. Software: Practice and Experience **53**(3), 856–892 (2023). DOI 10.1002/spe.3170
19. Vaithilingam, P., Zhang, T., Glassman, E.L.: Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In: Proc. of CHI'22, pp. 332:1–332:7. ACM (2022). DOI 10.1145/3491101.3519665
20. Weyssow, M., Sahraoui, H.A., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Software and Systems Modeling **21**(3), 1071–1089 (2022). DOI 10.1007/s10270-022-00975-5