2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS) | 979-8-3503-2480-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/MODELS58315.2023.00037

Automated Domain Modeling with Large Language Models: A Comparative Study

Kua Chen[*]
Electrical and Computer Engineering
McGill University
Montreal, Canada

Yujing Yang[*]
Electrical and Computer Engineering
McGill University
Montreal, Canada

Boqi Chen[*]
Electrical and Computer Engineering
McGill University
Montreal, Canada

José Antonio Hernández López[*]
DIS
University of Murcia
Murcia, Spain

Gunter Mussbacher
Electrical and Computer Engineering
McGill University
Montreal, Canada

Dániel Varró
Linköping University
McGill University
Linköping, Sweden / Montreal, Canada

***Abstract*—Domain modeling is an essential part of software engineering and serves as a way to represent and understand the concepts and relationships in a problem domain. Typically, software engineers interpret the problem description written in natural language and manually translate it into a domain model. Domain modeling can be time-consuming and highly depends on the expertise of software engineers. Recently, Large Language Models (LLMs) have exhibited remarkable ability in language understanding, generation, and reasoning. In this paper, we conduct a comprehensive, comparative study of using LLMs for fully automated domain modeling. We assess two powerful LLMs, GPT3.5 and GPT4, employing various prompt engineering techniques on a data set containing ten diverse domain modeling examples with reference solutions created by modeling experts. Our findings reveal that while LLMs demonstrate impressive domain understanding capabilities, they are still impractical for full automation, with the top-performing LLM achieving F_1 scores of 0.76 for class generation, 0.61 for attribute generation, and 0.34 for relationship generation. Moreover, the F_1 score is characterized by higher precision and lower recall; thus, domain elements retrieved by LLMs are often reliable, but there are many missing elements. Furthermore, modeling best practices are rarely followed in auto-generated domain models. Our data set and evaluation provide a valuable baseline for future research in automated LLM-based domain modeling.**

***Index Terms*—domain modeling, large language models, few-shot learning, chain-of-thought prompting, prompt engineering**

I. Introduction

Context: Domain modeling is a core process in software engineering that builds a domain model from various sources of information, including documents, stakeholder interactions, etc. This process is typically performed manually by software engineers, which can be time-consuming and highly dependent on human expertise. To mitigate this intensive manual effort, many approaches aim to automate this process [1]–[4]. However, these approaches have one or two main limitations: (1) some level of human interaction is required during generation, or (2) domain elements are extracted at the sentence level without considering the interaction between sentences. As such, no existing approaches can perform fully automated domain modeling while considering textual specifications as a whole without breaking them into sentences.

Problem statement: In this paper, we aim to address the problem of *fully automated* domain modeling. Given a textual description in its entirety, our aim is to derive a complete domain model *without any human interaction*.

Existing automated domain modeling approaches rely on rule-based techniques for the generation, while the machine learning models used are rather simple. One reason behind this is the lack of properly labeled training examples.

Recently, with rapid advances in auto-regressive language modeling, large language models (LLMs) have shown powerful generalizability to tasks beyond natural language processing [5]. LLMs can perform different tasks without supervised training on the specific task using carefully designed input (called prompt). Using different *prompt engineering* [6] techniques, LLMs can achieve impressive performance on different tasks by only using a few labeled examples in the prompt. Such advances raise the natural question of whether these LLMs can be used to *fully automate* domain modeling.

A domain model can be expressed by graphical or textual modeling languages such as UML [7], Umple [8], and Ecore [9]. These languages have strict syntax rules for representing various modeling elements. However, because LLMs are trained on a large corpus of text, it is difficult to constrain them to adhere to such strict syntax, particularly when modeling examples may infrequently appear in the training set. Therefore, an appropriate input prompt is necessary to instruct LLMs to generate outputs in the desired format.

Another challenge in automated domain modeling is the lack of proper evaluation methods. The same domain concept may be represented differently by different solutions. While

[*] All four authors contributed equally to this research.

Partially supported by grant TED2021-129381B-C22 (SATORI project) funded by MCIN/AEI/10.13039/501100011033, a FPU grant funded by the Universidad de Murcia, the FRQNT-B2X project (file number: 319955), IT30340 Mitacs Accelerate, and the Wallenberg AI, Autonomous Systems and Software Program (WASP), Sweden

some existing approaches can compare two domain models automatically [4], [10]–[13], they either only provide an imprecise estimation or only work under restricted situations.

Objectives: Our paper aims to evaluate to what extent domain modeling can be *fully* automated using an LLM *without* supervised training. In particular, we evaluate how the expressiveness of LLMs and different prompt engineering techniques can affect the quality of generated domain models. Furthermore, we aim to identify the advantages and limitations of current LLMs for fully automated domain modeling.

Contribution: Given a textual domain description, we present a novel approach[1] for fully automated domain modeling using different LLMs and prompt engineering techniques. The specific contributions of this paper are the following:

- We formulate the automated domain modeling problem as a text generation problem and propose a fully automated domain modeling pipeline that relies on an LLM.
- We provide a new data set for evaluating automated domain modeling including ten diverse modeling examples with reference solutions created by modeling experts.
- Our framework provides the domain model in a textual format suitable for various prompt engineering methods.
- We provide a semantic scoring technique and a detailed comparative evaluation of different configurations to assess the quality of auto-generated domain models.
- We conduct experiments with two LLMs, GPT3.5 [14] and GPT4 [15] with various prompt engineering methods on the newly created data set to evaluate the generated models with quantitative and qualitative criteria.

Added value: To our best knowledge, our paper is the first to use LLMs for fully automatically generating domain models from problem descriptions without supervised training. Furthermore, we carry out comparative experiments involving three different groups of prompt engineering techniques and assess the generated domain models, demonstrating the current advantages and limitations of LLMs in automated domain modeling. Our paper provides insightful guidelines for adapting LLMs in the modeling process. We also identify common fault patterns in the models generated by LLMs and provide suggestions for future improvements.

II. Background

A. Domain modeling

In domain modeling, engineers typically convert a textual domain specification into a domain model represented as a class diagram [7]. Domain modeling is a challenging and time-consuming task that requires expertise and experience. A domain model uses a subset of class diagram concepts to capture essential elements of a domain, their attributes, and their relationships but does not cover elements related to a more detailed design (e.g., operations and interfaces).

Example 1: Figure 1 (left) shows the H2S domain model, a delivery and pickup system, along with an excerpt of its problem description. This example covers the key concepts of domain models such as classes and enumerations (e.g., Person and ItemCategory, respectively), attributes (e.g., name), and three types of relationships among classes: a basic relationship called association (e.g., between Resident and Item), a whole-part relationship called composition (e.g., between Volunteer and Date), and an is-a relationship called generalization (e.g., between Item and FoodItem). Classes may be abstract (i.e., they cannot be instantiated). Multiplicities are specified for associations/compositions and indicate how many instances of one class may be related to instances of the other class (e.g., 0..1, 0..*). Role names may be specified for the classes in an association/composition (e.g., pickUpRoute). Compositions may also be specified by placing classes inside a composite class (e.g., H2S). In that case, multiplicities are shown next to the name of the contained class (e.g., Person*). Association classes and n-ary associations are excluded since they are not always supported (e.g., Ecore [9] does not support them).

B. Large language models (LLMs)

Language modeling is a traditional task in natural language processing that aims to estimate the conditional probability of a sequence of tokens. Recently, large language models (LLMs) have gained significant attention for this task. LLMs use deep neural networks, typically with transformer architecture [16], to estimate this probability distribution.

Given a sequence of tokens $s = \{s_1, s_2, ..., s_{k-1}\}$, LLMs estimate the conditional probability of the next token $P(s_k|s_1, ..., s_{k-1})$. These models are typically used for text generation through auto-regression. Specifically, in each time step, the LLM predicts a new token that is added to the input.

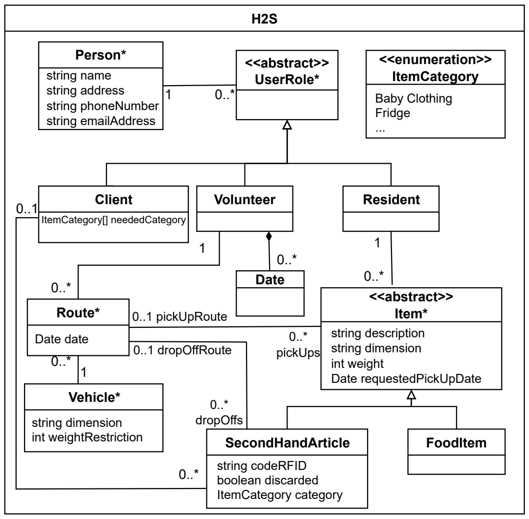
As the scale of LLMs (i.e., the number of parameters) increases, some models can be fine-tuned to perform other specific tasks beyond text generation [17].

C. Fine-tuning and prompting

a) Prompt-based learning: Fine-tuning of an LLM involves updating some of the parameters to fit a given task and a given data set [18]. This fine-tuning approach enhances the generalization ability of the trained models [18] but has two major drawbacks: (1) the training process can be computationally expensive, and (2) some tasks may not have sufficient training data for fine-tuning. Recently, LLMs have been used as few-shot learners [6], where the training examples are used as input rather than parameter updates. As an alternative to fine-tuning, prompt-based learning methods [19] involve constructing examples used as input to LLMs.

Given an input and a set of examples, a typical prompt-based learning method consists of three steps [19]: (1) transform the input and the examples into task context using a template that contains unfilled slots (a process known as *prompt engineering*), (2) fill the slots with an LLM to generate an output text, and (3) extract the final output from the output text using a post-processor. In this approach, the parameters of an LLM are fixed. Depending on the prompt engineering strategy, either a small number of labeled examples are needed, or no labeled examples are required.

[1]Artifact DOI: 10.5281/zenodo.8118642



Textual Domain Description: The Helping Hand Store (H2S) collects second hand articles and non-perishable foods from residents of the city and distributes them to those in need. H2S operates in many cities, but each location is run independently. To increase the number of items available for distribution, H2S is seeking to offer a Pickup and Delivery Service to its customers, which would allow a resident to schedule a pickup of items from a street address online at the H2S website.

A resident enters a name, street address, phone number, optional email address, as well as a description of the items to be picked up. H2S has a fleet of pickup vehicles, which it uses to collect items from residents. A pickup route for that day is determined for each vehicle for which a volunteer driver is available. Volunteer drivers indicate their available days on the H2S website. The route takes into account the available storage space of a vehicle and the dimensions and weights of scheduled items. A scheduled pickup may occur anytime between 8:00 and 14:00. After completing all scheduled pickups, the driver drops off all collected second hand articles at H2S's distribution center. Non-perishable foods, on the other hand, are directly dropped off at the food bank, which then deals with these items without further involvement from H2S ...

Enumerations:
ItemCategory(Baby_Clothing, Fridge, ...)
Classes:
H2S()
Person(string name, string address, string phoneNumber, string emailAddress)
abstract UserRole()
Client(ItemCategory[] neededCategory)
Volunteer()
Resident()
Date()
FoodItem()
SecondHandArticle(string codeRFID, boolean discarded,ItemCategory category)
Vehicle(string dimension, int weightRestriction)
abstract item(string description, string dimension, int weight, Date requestedPickUpDate)
Route(Date date)

Relationships:
1 H2S contain * Item
1 H2S contain * Vehicle
1 H2S contain * Route
1 H2S contain * UserRole
1 H2S contain * Person
1 Volunteer contain * Date
Client inherit UserRole
Volunteer inherit UserRole
Resident inherit UserRole
SecondHandArticle inherit Item
FoodItem inherit Item
1 Person associate * UserRole
0..1 Client associate * SecondHandArticle
1 Volunteer associate * Route
1 Vehicle associate * Route
0..1 Route associate * Item
0..1 Route associate * SecondHandArticle
1 Resident associate * Item

Fig. 1: Example domain model (left) and its textual representation (bottom right) with the problem description (top right)

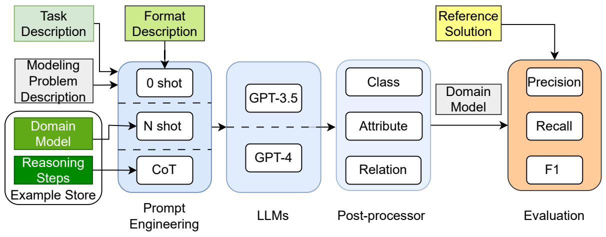


Fig. 2: Overview of automated domain modeling with LLMs

b) Prompt engineering: While many prompt engineering approaches exist for LLMs [19], here we focus on three widely used techniques. The **zero-shot** technique operates with no labeled examples, and task context is constructed using the task input, along with the textual instruction of the task. For example, in text classification, the task context can be "classify the following text input into categories: c1, c2..." along with the text to be classified. The **N-shot** technique adds N labeled examples to the task context in addition to the task description and input. However, using exclusively the input/output pair in the examples may not be sufficient for the LLM to learn a complex task involving multiple steps to solve. The **chain-of-thought (CoT)** technique [20] mitigates this issue by including a list of reasoning steps in the examples provided to the LLM. This technique allows the LLM to generate reasoning along with the task output and also improves the performance of LLMs on many benchmarks compared with N-shot techniques [20].

III. Approach

A. Overview

a) Problem formulation: Given a textual description in natural language, the *domain modeling* task consists of generating a domain model. More formally, let d be a domain specification with underlying ground truth domain model m; the problem of domain model generation is to define a generator $\mathtt{f}$ with $m' = \mathtt{f}(d)$, where m' is the generated domain model. Ideally, the generated model should be *similar* to the ground truth domain model, i.e., $m' \cong m$. In this paper, domain modeling is handled as a *text generation* problem. Specifically, the function $\mathtt{f}$ is LLM-based and the output of $\mathtt{f}$ is a textual description of m'.

b) Architecture: Figure 2 depicts the overall architecture of using an LLM for domain modeling. To generate a domain model, we provide a *task description*, a *modeling problem description* (domain specification), a *format description* (i.e., the expected output format for the domain model), examples of *domain modeling* descriptions with reference models, and the *reasoning steps* involved in the example as the task context to the LLM. In the prompt engineering stage, we employ three techniques: zero-shot, N-shot, and chain-of-thought (CoT).

Next, each prompt is fed into an LLM. We cover two different LLMs: GPT-3.5 and GPT-4. The LLM accepts text prompts as input and generates a text-based description of the domain model based on the provided format description. Next, a post-processor extracts classes, attributes, and relationships from the output of the LLM. The post-processor uses rule-based methods to parse and extract relevant results from the LLM's response to create the final domain model which is evaluated using the procedure described in Section IV.

B. Domain model representation

Instead of depicting a domain model using a graphical view or a specific modeling language, we present the domain model in structured natural language to avoid any biases caused by errors in modeling languages and to focus on the modeling capabilities of LLMs.

Figure 3 presents our domain model representation in Extended Backus-Naur Form (EBNF) format. The specification defines enumeration and two types of classes, regular and abstract, with the latter indicated by the keyword *abstract*.

```
<class-diagram> ::= [<enumerations>] <classes> <relationships>
<enumerations> ::= "Enumerations: " (<enumeration>)+
<enumeration> ::= <string> "(" <literals> ")"
<literals> ::= <string> | <string> ", " <literals>
<classes> ::= "Classes: " (<class>)+
<class> ::= ["abstract"] <string>"(" [<attributes>] ")"
<attributes> ::= <attribute> | <attribute> ", " <attributes>
<attribute> ::= <type>["[]"] <string>
<relationships> ::= "Relationships: " [<composition>]* [<inheritance>]* [<association>]*
<composition> ::= <mul> <string> "contain" <mul> <string>
<inheritance> ::= <string> "inherit" <string>
<association> ::= <mul> <string> "associate" <mul> <string>
<type> ::= <string>
<mul> ::= "*" | <num> | <num>".."("*"|<num>)
```

Fig. 3: EBNF format of a domain model

Enumeration literals are specified in brackets for each enumeration, while attributes (if any) can be multi-valued indicated by square brackets. Keywords identify three relationship types between two classes: *associate* for an association relationship, *inherit* for a generalization relationship, and *contain* for a composition relationship. Multiplicity is required and positioned before each class name for association and composition. We ignore the role names for associations and compositions in the representation for simplicity. Figure 1 (bottom right) shows the H2S example following the EBNF format.

C. Prompt engineering

In the next step, we take a problem description as input and design a prompt to describe the domain model generation task with three settings in the prompt design: zero-shot, N-shot, and chain-of-thought. Figure 4 illustrates an example for each prompting method.

Zero-shot learning: In the context of zero-shot learning, the LLM is not provided with any examples, and instead, it receives only natural language instructions describing the task. To facilitate this, we design a prompt that consists of a *task description*, a *format description*, and a target *problem description* for generating a domain model. The task description serves the purpose of defining the objective of the task for the LLM, which is to generate classes, attributes, and relationships of a domain model in natural language format, based on the given problem description.

The first example in Figure 4 shows a prompt based on zero-shot learning for the LabTracker problem description, a medical examination system. The format description specifies the desired output format using textual description of the EBNF format in Figure 3. The modeling problem description provides information about the target modeling domain, including the target system and its components and relationships.

N-shot learning: In the case of N-shot learning, N examples are provided to the LLM as input-output pairs to learn both the format and content of the desired output. To facilitate this, we design a prompt that includes a *task description*, N *examples* of the problem description and solution, and a modeling *problem description* for the model to generate.

The task description and problem description are the same as in the zero-shot learning setup. However, in N-shot learning, instead of a format description, we provide a detailed example in the defined format. The example consists of two parts: a problem description of an example domain and a reference domain model in textual representation for the example. One sample prompt for N-shot learning to generate a domain model for LabTracker, with the transportation system BTMS as an example, is shown as the second example in Figure 4.

Chain-of-thought: In this setting, the prompt consists of a *task description*, chain-of-thought *reasoning steps*, and a modeling *problem description*. In previous N-shot settings, the problem description and solution were provided separately, without a clear indication of how the solution was derived from the description. To address this issue, we use chain-of-thought [20] prompting, which involves presenting a coherent series of intermediate steps leading to the solution. Instead of providing the description and solution separately, we integrate them by appending relevant modeling elements in the solution to each sentence, providing reasoning steps at the sentence level. This approach offers a more comprehensive illustration of how each element is derived from the text.

The last example in Figure 4 demonstrates a prompt that uses CoT to generate a domain model for LabTracker, with pairs of problem description sentences / relevant solutions for H2S.

D. Post-processing

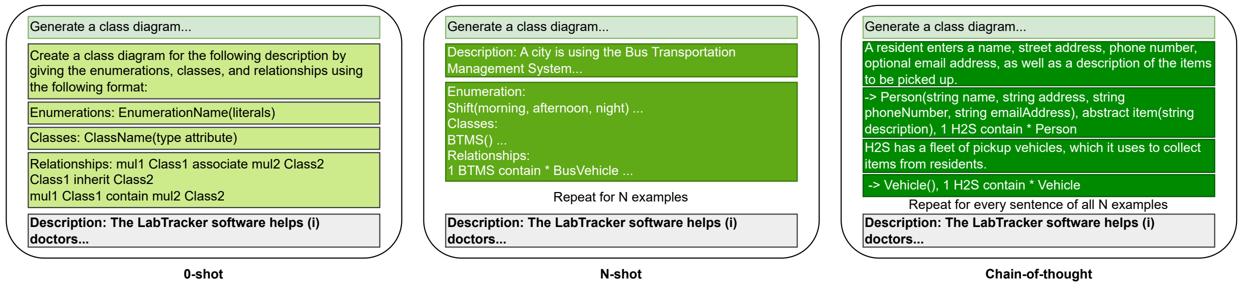
The post-processor takes the natural language output of LLMs and converts it into a domain model format using a rule-based method. The output of LLMs generally follows the structured natural language format defined before, except for certain parts especially in zero-shot setups. In cases where the generated text does not conform to the specified format, the post-processor adapts the output to the desired format. In particular, some of the generated relationships may have an association name instead of the three relationship keywords, and we treat all such relationships as *associations*. Additionally, some of the generated attributes may not have a data type, and they are set to a default type of string.

IV. Domain Modeling Evaluation

A. Evaluation procedure

This paper focuses on an ideal domain modeling scenario where the problem description is well-constructed and clear. We gather ten problem descriptions written in English (Table I) from an undergraduate-level, model-driven programming course. The examples are taken from projects or exams in the course and have been used across multiple course offerings, covering a wide range of domains. Each example has a domain model developed by modeling experts as a reference solution, which is originally used to grade student models in the course. Given a problem description, we evaluate the quality of the generated domain models by comparing them with the reference solution (model).

Domain model evaluation is a difficult task as there are multiple acceptable solution models for the same problem description. Moreover, the automated comparison of two domain models is also non-trivial. For example, there are many ways to



Generate a class diagram...

Create a class diagram for the following description by giving the enumerations, classes, and relationships using the following format:

Enumerations: EnumerationName(literals)

Classes: ClassName(type attribute)

Relationships: mul1 Class1 associate mul2 Class2
Class1 inherit Class2
mul1 Class1 contain mul2 Class2

Description: The LabTracker software helps (i) doctors...

0-shot

Generate a class diagram...

Description: A city is using the Bus Transportation Management System...

Enumeration:
Shift(morning, afternoon, night) ...
Classes:
BTMS() ...
Relationships:
1 BTMS contain * BusVehicle ...

Repeat for N examples

Description: The LabTracker software helps (i) doctors...

N-shot

Generate a class diagram...

A resident enters a name, street address, phone number, optional email address, as well as a description of the items to be picked up.

-> Person(string name, string address, string phoneNumber, string emailAddress), abstract item(string description), 1 H2S contain * Person

H2S has a fleet of pickup vehicles, which it uses to collect items from residents.

-> Vehicle(), 1 H2S contain * Vehicle

Repeat for every sentence of all N examples

Description: The LabTracker software helps (i) doctors...

Chain-of-thought

Fig. 4: Example prompts for three prompting methods

name a class or an attribute where the semantics is the same. While there are approaches to compare two domain models automatically [4], [10]–[13], they either provide an imprecise estimation or work under carefully selected situations.

Due to the limitation of automated evaluators, we choose to manually evaluate the generated domain models to precisely measure the modeling ability of LLMs. To ensure consistency and fairness in evaluation, we conduct the manual evaluation using a consensus approach and the process is summarized in Figure 5. Two authors first evaluate the same two generated models separately, then both authors compare their results, resolve differences and come to a mutual agreement on an evaluation scheme. Afterwards, two authors start the first evaluation round. For consistency, they each independently evaluate half of all examples in all settings. After Round 1, a calibration phase is performed where all authors review the grading and suggest improvements for a second round of evaluation. Finally, the same two authors start the second evaluation round.

We calculate *Cohen's d-score* [21] between the first and second evaluation results and get an effect size of 0.10 (small effect), which suggests the two rounds of evaluation only have minor differences. We use the results from the second round of evaluation as the final result.

B. Evaluation scheme

The first step in evaluating each modeling element (i.e., class, attribute, and relationship) is to determine whether it has a direct counterpart in the reference model. To compare a domain model with a reference domain model, we use an evaluation scheme. Motivated by existing work [10], we categorize each modeling element into one of four categories as shown in Table II

- *c1* Category *c1* includes all domain elements that have an exact match with the elements of the same type in the reference model. It is important to note that this match is based on semantics rather than string comparison.
- *c2* Elements in category *c2* are matched with an element that may not be the same type in the reference model but are semantically equivalent to the matched element.
- *c3* Category *c3* is for elements that only partially match the element in the reference model. This category also includes elements without a match in the reference model due to another incorrect design decision (i.e., consequential mistakes) that are otherwise correct.
- *c4* Category *c4* captures incorrect elements that do not match any elements in the reference model.

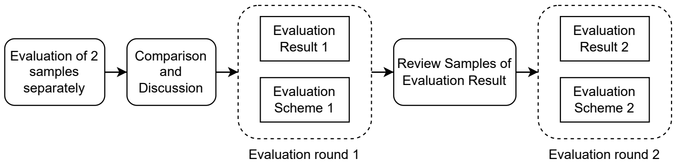


Fig. 5: Flowchart of the evaluation process

We categorize all elements in *both* the generated and reference model to compute precision and recall of the generated model.

Example 2: The last column of Table II shows examples of each category for the H2S domain, separated by a colon. Left of the colon presents the element from the reference model, and the right is the element to be evaluated. To ensure conciseness, we omit attributes of certain classes.

For *c1*, the elements are exactly the same or semantically equivalent to the reference model. For example, both the reference model and generated model contain the H2S class. Additionally, the Person class of the reference model is matched with the User class, as both capture the same domain concept. Elements in *c2* are still equivalent, but may not be of the same type. For example, the SecondHandArticle class has a boolean attribute *discarded* in the reference solution, while the generated model includes an enumeration Status with two literals and adds an enumeration attribute to the class.

Category *c3* encompasses elements that only partially match the reference model. In the H2S domain model, for example, there is a relationship stating that 0 or 1 Route is associated with 0 or many Items. There are two possible cases for the association to fall into this category: (1) Route is marked as non-optional, leading to a partial match of the relationship with the reference model; (2) the generated model may not have an abstract superclass corresponding to Item; thus Route is associated with FoodItem instead. In the latter case, although the association does not match, it is correct given the superclass mistake in the generated model, and we consider it as *c3*.

Elements in category *c4* are without a match. In the H2S example, the abstract UserRole class in the reference model has no equivalent modeling element in the generated model.

Name	BTMS	H2S	LabTracker	CelO	TeamSports	SHAS	OTS	Block	Tile-O	HBMS
Domain	Transportation	Delivery	Medical	Social	Sports	Smart Home	Education	Game	Game	Management
# of classes	7	13	16	13	16	23	16	15	18	18
# of attributes	11	18	43	23	24	26	25	30	19	32
# of relationships	9	18	22	22	20	27	19	24	21	22

TABLE I: Collected modeling examples and their domains

Category	**Description**	**Examples**
c1	Direct match	H2S():H2S(), Person(...):User(...)
c2	Semantically equivalent	SecondHandArticle(boolean discarded,...): SecondHandArticle(Status status,...), Status(AVAILABLE, DISCARDED)
c3	Partial match	0..1 Route associate * Item: (1) 1 Route associate * Item (2) 0..1 Route associate * FoodItem
c4	No match	abstract UserRole():(No role class)

TABLE II: Scheme for comparing two domain models

To evaluate the modeling elements (classes, attributes, and relationships), we use a modified version of the precision, recall, and F_1 score metrics. Unlike binary scoring, we assign 0.5 points to generated elements that partially match the reference solution. Specifically, elements in the first two categories are awarded 1 point, while elements in *c3* receive 0.5 points, and those in *c4* receive 0 points. The scoring function S for an element x can be expressed formally as follows:

$$S(x) = \begin{cases} 1 & \text{if } x \text{ is in c1 or c2} \\ 0.5 & \text{if } x \text{ is in c3} \\ 0 & \text{if } x \text{ is in c4} \end{cases} \tag{1}$$

C. Evaluation criteria

We aim to evaluate the quality of the output domain model. To do so, we consider the precision, recall, and F_1 scores over the classes, relationships, and attributes. Precision measures the overall correctness of generated modeling elements. For example, let $\mathcal{C}$ be the set of all classes and enumerations in the generated model of size $m = |\mathcal{C}|$, the precision of classes in generated models can be expressed as

$$Precision_{\mathcal{C}} = \frac{\sum_{i=0}^{i=m} S(\mathcal{C}_i)}{m}, \tag{2}$$

where S is the scoring function defined in Equation 1.

Recall measures the degree of the reference model covered by the generated model. For example, let $\mathcal{C}_{gt}$ be the set of classes and enumerations in the *reference model* (ground truth) of size $n = |\mathcal{C}_{gt}|$, the recall of classes can be expressed as

$$Recall_{\mathcal{C}} = \frac{\sum_{i=0}^{i=n} S(\mathcal{C}_i)}{n}. \tag{3}$$

Finally, we use the classical F_1 score definition:

$$F_{1\mathcal{C}} = \frac{2 \times Precision_{\mathcal{C}} \times Recall_{\mathcal{C}}}{Precision_{\mathcal{C}} + Recall_{\mathcal{C}}}. \tag{4}$$

Metrics for attributes or relationships can be computed by substituting $\mathcal{C}$ with the set of attributes or relationships.

V. Experiments

The aim of this section is to assess the capability of LLMs in domain modeling, identify areas of the domain modeling task where these models may encounter challenges, provide insights on the most effective prompt engineering approach for generating domain models using such LLMs, and compare several LLMs. More concretely, we aim to investigate the following three research questions (RQ):

1: How do LLMs perform in automated domain modeling?
2: What is the effect of using different prompts?
3: Which one of the assessed LLMs performs best?

We begin by describing the experimental settings in Section V-A. Subsequently, we tackle each one of the three research questions and report findings in Sections V-B,V-C, and V-D. Finally, we provide an in-depth analysis and discussion of the results including qualitative results in Section V-E and state threats to validity in Section V-F.

A. Experimental settings

Large language models. We experiment with two GPT-3.5 models (`Davinci` and `Turbo`) and one GPT4 model (`GPT4`).

- `Davinci` is the *text-davinci-003* model from OpenAI GPT3.5 API [22]. It performs text completion tasks by receiving input text and producing output text.
- `Turbo` is the *gpt-3.5-turbo* via the OpenAI chat completion API. It uses a similar model architecture as Davinci but it is optimized for chat completion. We adapt our prompts to become chat-like for this model.
- `GPT4` is the *GPT-4* model recently released by OpenAI. We use the web interface for accessing GPT4 via the ChatGPT website [23] because the GPT4 API is not yet publicly available to all developers.

Prompt settings. We experiment with one `0-shot`, three N-shot (`1-shot-btms`, `1-shot-h2s`, and `2-shot`), and one chain-of-thought (`CoT`) prompt setting as described in Section III-C. We select one simple domain (BTMS) and a complex domain that contains many best practices (H2S) as the two examples. `1-shot-btms` uses a prompt with the BTMS and its reference domain model, while `1-shot-h2s` uses the H2S. Furthermore, `2-shot` combines the BTMS and H2S and `CoT` uses H2S as the example.

Test set. To evaluate each LLM and each prompt setting, we use the descriptions and the associated ground truth domain models of examples in Table I. We remove the domain models of BTMS and H2S from the test set as they are introduced in the input prompts; thus, we consider a test set of eight samples. Each example represents a software system, and the domain models require the use of some advanced modeling patterns

TABLE III: Average performance scores for each setting and modelling element of `Davinci`

	Class			Attribute			Relationship		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
`0-shot`	0.89 ± 0.10	0.39 ± 0.06	0.54 ± 0.07	0.57 ± 0.20	0.37 ± 0.14	0.44 ± 0.16	0.28 ± 0.19	0.10 ± 0.06	0.14 ± 0.09
`1-shot-btms`	0.96 ± 0.06	0.48 ± 0.09	0.63 ± 0.08	0.73 ± 0.15	0.31 ± 0.14	0.42 ± 0.15	0.55 ± 0.16	0.24 ± 0.10	$\mathbf{0.33 \pm 0.12}$
`1-shot-h2s`	0.95 ± 0.06	0.46 ± 0.07	0.62 ± 0.07	0.65 ± 0.15	0.31 ± 0.15	0.41 ± 0.18	0.51 ± 0.14	0.18 ± 0.09	0.26 ± 0.12
`2-shot`	0.91 ± 0.06	0.50 ± 0.06	$\mathbf{0.64 \pm 0.05}$	0.67 ± 0.16	0.38 ± 0.13	$\mathbf{0.48 \pm 0.14}$	0.54 ± 0.17	0.22 ± 0.08	0.31 ± 0.11
`CoT`	0.89 ± 0.11	0.47 ± 0.06	0.61 ± 0.07	0.55 ± 0.16	0.33 ± 0.14	0.39 ± 0.14	0.35 ± 0.22	0.12 ± 0.08	0.17 ± 0.11

TABLE IV: Average performance scores for each setting and modelling element of `Turbo`

	Class			Attribute			Relationship		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
`0-shot`	0.87 ± 0.09	0.41 ± 0.11	0.55 ± 0.11	0.52 ± 0.22	0.38 ± 0.20	0.43 ± 0.20	0.26 ± 0.11	0.10 ± 0.04	0.14 ± 0.06
`1-shot-btms`	0.95 ± 0.05	0.51 ± 0.13	0.66 ± 0.12	0.64 ± 0.19	0.33 ± 0.18	0.42 ± 0.17	0.36 ± 0.23	0.19 ± 0.11	0.24 ± 0.14
`1-shot-h2s`	0.85 ± 0.09	0.56 ± 0.09	0.67 ± 0.08	0.54 ± 0.15	0.37 ± 0.14	0.43 ± 0.14	0.35 ± 0.14	0.20 ± 0.05	0.24 ± 0.07
`2-shot`	0.87 ± 0.05	0.56 ± 0.09	$\mathbf{0.68 \pm 0.08}$	0.56 ± 0.16	0.41 ± 0.14	0.46 ± 0.11	0.39 ± 0.14	0.23 ± 0.08	$\mathbf{0.29 \pm 0.10}$
`CoT`	0.88 ± 0.13	0.44 ± 0.07	0.58 ± 0.08	0.67 ± 0.23	0.38 ± 0.21	$\mathbf{0.46 \pm 0.19}$	0.33 ± 0.23	0.12 ± 0.08	0.17 ± 0.11

TABLE V: Average performance scores for each setting and modelling element of `GPT4`

	Class			Attribute			Relationship		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
`0-shot`	0.89 ± 0.05	0.54 ± 0.05	0.67 ± 0.04	0.64 ± 0.21	0.53 ± 0.13	0.57 ± 0.14	0.44 ± 0.11	0.18 ± 0.03	0.25 ± 0.05
`1-shot-btms`	0.93 ± 0.05	0.64 ± 0.07	$\mathbf{0.76 \pm 0.06}$	0.72 ± 0.13	0.54 ± 0.20	0.58 ± 0.16	0.50 ± 0.14	0.26 ± 0.07	$\mathbf{0.34 \pm 0.08}$
`1-shot-h2s`	0.83 ± 0.09	0.60 ± 0.05	0.70 ± 0.05	0.66 ± 0.19	0.54 ± 0.18	0.58 ± 0.18	0.51 ± 0.21	0.25 ± 0.11	0.33 ± 0.14
`2-shot`	0.94 ± 0.06	0.63 ± 0.09	0.75 ± 0.05	0.70 ± 0.18	0.56 ± 0.14	$\mathbf{0.61 \pm 0.13}$	0.45 ± 0.20	0.27 ± 0.11	0.34 ± 0.14
`CoT`	0.87 ± 0.09	0.55 ± 0.06	0.67 ± 0.06	0.64 ± 0.13	0.45 ± 0.19	0.50 ± 0.13	0.42 ± 0.13	0.21 ± 0.06	0.27 ± 0.08

(e.g., abstraction-occurrence, player-role), but the actual patterns covered in various examples are different.

Statistical tests. When comparing two groups of samples through a statistical test, the Wilcoxon signed-rank test [24] is used as several variables of the data set do not follow a normal distribution based on the Shapiro-Wilk normality test. The significance level is set to 0.05.

B. RQ1: Performance of LLMs in domain modeling

RQ1 aims to evaluate how LLMs perform in domain models generation. We are interested in how far the LLMs are from a perfect performance score for this task. This will show if there is room for improvement and whether the automated domain model generation problem is solved with LLMs. Furthermore, we carry out a more fine-grained analysis by highlighting with which modeling aspects the LLMs struggle. Particularly, we compare the performances of the LLMs when recovering classes, attributes, and relationships; and we study for which evaluation metrics (precision or recall) the LLMs struggle.

Results. Tables III, IV, and V show the average precision, recall, and F_1 scores for each LLM, setting, and type of modeling element. It can be observed that the best average F_1 scores are obtained by `GPT4` and they are the following:

- Class modeling element: the best F_1 score is 0.76, achieved by `1-shot-btms`.
- Attribute modeling element: the best F_1 score is 0.61, achieved by `2-shot`.
- Relationship modeling element: the best F_1 score is 0.34, achieved by `1-shot-btms`.

From these results, we can also infer that LLMs generate classes from descriptions better than attributes and they generate attributes better than relationships. Using the Wilcoxon signed-rank test, we compare the F_1 scores and see if these differences are significant. After applying the pairwise tests for each LLM and setting, we report the number of patterns in Table VI. The pattern that mostly occurs is *Class $>$ Attribute $>$ Relationship* (8 out of 15). Furthermore, there are important differences between the F_1 scores of each pair of modeling elements. Particularly, the LLMs struggle when generating relationships. Comparing the average relationship F_1 score with the other two modeling elements in Tables III, IV, and V, one can notice the poor performance of LLMs in extracting relationships. To better illustrate this, we show the boxplots of the F_1 scores for each type of modeling element in Figure 6 associated with `GPT4` using the setting `1-shot-btms`.

TABLE VI: Patterns found when applying pairwise tests of types of modeling elements for each LLM and setting (the symbol $>$ means "statistically better than" and $\approx$ means that no statistical difference is found)

Pattern	Number of occurrences
Class $>$ Attribute $>$ Relationship	8
Class $\approx$ Attribute $>$ Relationship	5
Class $>$ Attribute $\approx$ Relationship	2

TABLE VII: Positive tests when comparing the precision and the recall scores

Type of modeling element	Positive tests / Total tests
Class	15/15
Attribute	9/15
Relationship	15/15

Finally, we study for which metric, precision or recall, the LLMs struggle more. From the figures presented in Tables III, IV, and V, it seems clear that LLMs obtain a higher precision score. We carry out a statistical test to compare the precision and recall for each LLM, setting, and type of modeling element and show the positive tests in Table VII listed by type of modeling element. We can observe that all

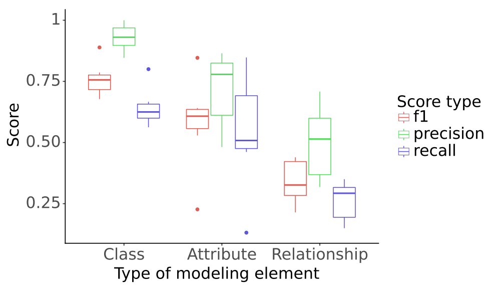


Fig. 6: F_1 score, precision, and recall boxplots associated with `GPT4` using the setting `1-shot-btms` ($x-$axis represents the type of modeling elements and $y-$axis the scores)

of the tests are positive in case of classes and relationships, and the majority of tests are positive in case of attributes. We also highlight that the recall of the LLMs is, in general, much lower than the precision (see Tables III, IV, and V). Thus, the recall is the one that negatively impacts the overall performance (i.e., the F_1 score). For instance, it can be observed that the recall score is much lower than the precision score for all modeling elements in `GPT4` with `1-shot-btms` setting (Figure 6).

Answer to RQ1. While LLMs demonstrate impressive domain understanding capability, they are still impractical for fully automated domain modeling. The top-performing LLM achieves F_1 scores of 0.76 for classes, 0.61 for attributes, and 0.34 for relationships. Moreover, LLMs struggle the most with identifying relationships (compared to classes and attributes). Altogether, domain elements generated by LLMs are often reliable but there are many missing elements.

C. *RQ2: Effects of prompt engineering*

The output of LLMs is influenced by the structure and content of the prompts. RQ2 investigates (1) if adding examples to the prompt helps the LLMs to produce better domain models and (2) if adding reasoning steps improves the performance of LLMs in domain modeling.

Effect of examples. We first use the `0-shot` setting as the baseline prompting method and it is compared with the two 1-shot settings (`1-shot-btms` and `1-shot-h2s`) and the `2-shot` setting. Concretely, given an LLM, a type of modeling element, and a few-shot prompting method, we run the Wilcoxon test to compare the F_1 score with respect to the baseline. We also compare the `2-shot` setting with the `1-shot-btms` and `1-shot-h2s` settings similarly to see if adding more examples is better than adding one.

Results. Tables VIII IX, and X show the $p-$values and the mean differences of the F_1 scores ($\mu - \mu_{\texttt{0-shot}}$) when comparing each few-shot prompt setting with the `0-shot` setting for each LLM. We observe statistically significant improvements when generating classes in `Davinci` and `GPT4`. Significant improvements also exist in relationships,

TABLE VIII: `Davinci` comparisons for each prompting method and for each type of model element (cell format: $p-$value/mean difference; bold means significant)

Setting	Class	Attribute	Relationship
`1-shot-btms`	**0.0391 / 0.0929**	0.4609 / -0.0244	**0.0156 / 0.1862**
`1-shot-h2s`	**0.0078 / 0.0789**	0.0781 / -0.0367	**0.0078 / 0.1206**
`2-shot`	**0.0225 / 0.1051**	0.4609 / 0.0353	**0.0078 / 0.1739**

TABLE IX: `Turbo` comparisons for each prompting method and for each type of model element (cell format: $p-$value/mean difference; bold means significant)

Setting	Class	Attribute	Relationship
`1-shot-btms`	0.0781 / 0.1097	0.3828 / -0.0112	0.0781 / 0.1066
`1-shot-h2s`	0.0547 / 0.1250	0.8438 / -0.0009	**0.0234 / 0.1087**
`2-shot`	0.0547 / 0.1345	0.5469 / 0.0250	**0.0078 / 0.1520**

particularly, in `Davinci` and `Turbo`. Regarding attributes, no significant improvement is observed in any of the LLMs.

We also study if there are differences between the `2-shot` setting and the `1-shot` settings. For each LLM and type of modeling element, we find no significant differences when comparing `2-shot` with both `1-shot` settings.

Effect of reasoning steps. To address this aspect, we compare the average F_1 scores of the `1-shot-h2s` and its `CoT` version for each type of modeling element and LLM.

Results. Table XI shows the mean differences of the F_1 scores ($\mu_{\texttt{CoT}} - \mu_{\texttt{1-shot-h2s}}$) for each LLM and type of modeling element. We can observe a performance decrease for almost all combinations. There is only one case where the performance difference is positive but it is not statistically significant.

Answer to RQ2. Adding examples to the prompt has positive effects on the performance of LLMs when retrieving classes and relations. However, we do not have enough evidence to claim that adding two samples to the prompt is better than adding one. Finally, adding reasoning steps may decrease the performance.

D. *RQ3: The best of the assessed LLMs*

RQ3 aims to find out the best LLM of the ones we have accessed. We compare the three LLMs across all settings

TABLE X: `GPT4` comparisons for each prompting method and for each type of model element (cell format: $p-$value/mean difference; bold means significant)

Setting	Class	Attribute	Relationship
`1-shot-btms`	**0.0234 / 0.0864**	0.7422 / 0.0168	0.0547 / 0.0849
`1-shot-h2s`	0.3828 / 0.0264	0.6406 / 0.0190	0.2500 / 0.0780
`2-shot`	0.0781 / 0.0765	0.3125 / 0.0416	0.1484 / 0.0841

TABLE XI: Mean F_1 differences ($\mu_{\texttt{CoT}} - \mu_{\texttt{1-shot-h2s}}$) for each LLM and type of modeling element

Type of modeling element	`Davinci`	`Turbo`	`GPT4`
Class	-0.0032	-0.0865	-0.0289
Attribute	-0.0144	0.0275	-0.0827
Relationship	-0.0890	-0.0695	-0.0558

TABLE XII: Each cell contains the model that achieves the highest average F_1 score (bold means that the comparison of that model with the other two is significant)

Setting	Class	Attribute	Relationship
0-shot	**GPT4**	**GPT4**	**GPT4**
1-shot-btms	GPT4	**GPT4**	GPT4
1-shot-h2s	GPT4	GPT4	GPT4
2-shot	**GPT4**	GPT4	GPT4
CoT	GPT4	GPT4	GPT4

and modeling elements. We report which LLM obtains the highest average F_1 score for each type of modeling element and prompt setting (a total of 15 comparisons).

Results. Table XII shows that GPT4 achieves the highest F_1 score for all settings and types of modeling element. The differences are statistically significant in only five cases and three of them correspond to the 0-shot setting.

Answer to RQ3. GPT4 achieves the highest average F_1 score in all cases with the strongest evidence found in the 0-shot setting, suggesting that GPT4's modeling ability is better compared to the other models.

E. Discussion

In this section, we discuss several aspects of our experiments and possible implications.

RQ1. The experiments show that LLMs are still not able to fully automate the domain modeling task. However, the results obtained are promising and there is still room for improvement. The fine-gradient analysis reveals that these generated domain models mainly fail to recover relationships and miss a lot of modeling elements from the description (low recall). Thus the focus of future work should be to improve these two aspects. One potential future direction is to make LLMs generate domain models multiple times for one problem description and aggregate all the domain models in a way that more modeling elements in the reference domain model can be covered. At the same time, given LLMs can often generate correct model elements, their behavior in an interactive modeling setting can also be studied. In this case, a human modeler can continuously provide feedback on the generated model to continuously improve the generated model.

RQ2. By answering the second research question, we provide insights on how to design prompts for domain modeling. Adding examples to the prompt helps to generate classes and relations but we do not find significant improvements when comparing the 2-example setting and the 1-example setting. One future direction could be to focus on devising a method to select good examples for the prompt. There is already existing work in the LLMs literature that deals with this problem [25]–[27]. We found that the CoT prompting technique may negatively impact performance. We believe the negative impact is caused by the fact that domain modeling tasks require full comprehension of the description and it is not natural to see it as a sequence of independent reasoning steps. Therefore, innovative prompting techniques designed for domain modeling are needed to further improve the performance.

RQ3. When comparing the performances of the LLMs, we have found evidence that the GPT4 model performs the best in the domain modeling task and this evidence is stronger in the 0-shot setting. This is because GPT4 has better language understanding skills (particularly, better comprehension skills) than the other two models assessed in the experiments [15].

Modeling best practices. We also investigate the performance of the LLMs qualitatively, i.e., to what an extent best modeling practices are used. Since GPT4 is found to have the best performance among the investigated LLMs, we analyze the 2-shot setting results from GPT4. We find that models generated by GPT4 tend to have a common anti-pattern that misidentifies a domain concept as an enumeration rather than a class, with nine cases found in all examples. However, we do not observe any enumeration that is misidentified as a class. We observe a similar scenario where relationships are treated as attributes in the generated models, with 17 occurrences in eight generated models. At the same time, there is only one case where an attribute is treated as a relationship in the generated models. Surprisingly, GPT4 does not generate any correct abstract class. Meanwhile, common best-practice modeling patterns, such as player-role and abstraction occurrence [28], rarely appear in the generated models. No complete player-role pattern or abstraction-occurrence pattern is found in any generated models, and only one player-role pattern is partially found. This result may indicate that LLMs struggle with complex modeling that requires modeling knowledge beyond the domain description. Thus, trying to inject such modeling knowledge into these LLMs can be an interesting future work.

F. Threats to validity

Internal validity. The output of an LLM can be slightly different for each run. We address this variation by experimenting on the generation of eight diverse domain examples. The manual evaluation is done by two authors, which may introduce bias. We mitigate this bias by having a consensus process. We also have multiple rounds of evaluations where other authors review the evaluation results. Meanwhile, the selections of weights in the scoring function may influence experiment results. We choose scoring methods widely used to evaluate university-level assignments. There are typically multiple ways to represent a domain model, which may influence the performance of LLMs. To address this issue, we define an independent (new) text-based domain model representation that is based on natural language and use it across all experiments.

External validity. Due to the lack of benchmark data sets for domain modeling with both problem description and reference solution, we curate a data set of ten modeling examples with reference models created by domain modeling experts. We skipped the $p-$value adjustment in the statistical test as the number of modeling examples is relatively small (eight per distinct group). Therefore, there is a higher risk of getting false positives in the statistical tests. Furthermore, we collect models from an undergraduate modeling course representing modeling in education scenarios. LLMs may perform differently if used in a different scenario or with larger models.

Construct validity. Our paper adapts metrics widely used for evaluating the generated domain models [1]–[4].

VI. Related Work

A. *Automated domain modeling.*

Many existing works on automated domain modeling utilize statistical methods or rule-based methods to directly derive complete domain modeling solutions like UML class diagrams [1]–[4] or provide modeling assistance and suggestions [29], [30] from textual descriptions in natural language.

Statistical methods emphasize using machine learning techniques to extract domain models. Burgueño et al. [29] design a framework to suggest new model elements for a given partially completed model, by using word embeddings to capture the lexical and semantic information from textual documents. Several other existing approaches also combine NLP and ML techniques to automate the model creation process [2], [4].

Rule-based methods include using hand-written grammatical templates and heuristics. An example of a rule-based method presents an algorithm with 23 heuristics to automatically identify model elements from user stories [3]. Another example of such rule-based methods is proposed by Herchi et al. [31]. The authors begin by using an NLP toolkit including a sentence splitter, tokenizer, and syntactic parser to decompose the input text and then use linguistic rules (e.g., *All nouns are converted to entity types*) to extract UML concepts.

Our paper proposes a statistical method for *fully* automated domain modeling using LLMs. We investigate the use of pre-trained generative LLMs, for domain modeling, with only a limited number of labeled examples and no extra training.

B. *Large language models for MDE.*

With the advancement of LLMs, various language models have been incorporated within model-based engineering (MDE), yielding significant improvements in various aspects of the development process.

Weyssow et al. [30] propose a learning-based approach to recommend relevant domain concepts to a modeler during a meta-modeling activity by training a deep-learning model (a RoBERTa model trained on thousands of independent metamodels). Chaaben et al. [11] propose an approach for model completion by generating related elements using GPT-3. They formulate examples using classes and their relationships with other classes and then create prompts with few-shot learning. They also rank the generated classes by frequency in multiple runs. Others discuss the potential of ChatGPT in software engineering [32], [33]. For example, Cámara et al. [33] present the use of ChatGPT to build UML class diagrams enriched with OCL constraints in an interactive mode.

Compared with existing approaches adapting LLMs to MDE, we investigate fully automated domain model generation tasks with pre-trained generative LLMs given problem descriptions and including classes, attributes, and relationships.

C. *Domain model evaluation.*

Evaluating generated domain models can be challenging due to the variety of model elements and intricate design patterns involved. Many approaches have been investigated to assess the resulting solution in comparison to the reference solution. Yang and Sahraoui [4] combine learning-based and rule-based approaches for the extraction of UML class diagrams from natural language software specifications. In their evaluation, they evaluate the result with reference solutions using exact matching, relaxed matching, and general matching over classes and relationships. Bien et al. [10] present an approach for automated grading of UML class diagrams, which uses a grading algorithm that uses syntactic, semantic, and structural matching between two class diagrams. Boubekeur et al. [13] propose an approach based on a simple heuristic and machine learning that helps categorize simple domain model submissions from students according to their quality. The system determines if submissions are above a quality threshold to assign them a letter grade. Singh et al. [12] introduce a Mistake Detection System (MDS) designed to identify errors and offer feedback to students by comparing their submissions with a solution. This system is able to detect a wide range of potential mistakes present in a submission.

In this paper, we use a carefully designed manual assessment of the generated domain models. In comparison to existing automated evaluation methods, this manual evaluation takes into account partially correct results and separately calculates scores for different types of domain elements, offering a more comprehensive evaluation of the model's performance.

VII. Conclusion

This paper presents an approach for fully automated domain modeling using Large Language Models (LLMs). Given a domain description, the system directly generates a textual representation of the domain model without any human interaction. We evaluate this approach using two powerful LLMs (GPT-3.5 and GPT-4) along with various prompt engineering techniques on a data set of ten domain examples with reference solutions created by modeling experts. Our results demonstrate that while the top-performing LLM, GPT-4, shows impressive domain understanding capability without explicit training, fully automating domain modeling remains impractical. Furthermore, LLM-generated domain elements exhibit high precision but low recall. Specifically, LLMs struggle the most with relationships. We also identify common fault patterns in the models generated by LLMs and provide suggestions for future improvements. Moreover, we note that advanced modeling best practices are rarely applied in generated models.

This study highlights the potential of LLMs for fully automated domain modeling and points to the challenges that must be addressed for full automation. Future work includes designing prompting methods designated for domain modeling, proposing a framework for the automated selection of examples used in prompting, and injecting explicit modeling knowledge into LLMs.

References

[1] J. Francŭ and P. Hnětynka, "Automated generation of implementation from textual system requirements," in *Software Engineering Techniques: Third IFIP TC 2 Central and East European Conference, CEE-SET 2008, Brno, Czech Republic, October 13-15, 2008, Revised Selected Papers 3*. Springer, 2011, pp. 34–47.

[2] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Machine learning-based incremental learning in interactive domain modelling," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2022, p. 176–186.

[3] M. Robeer, G. Lucassen, J. M. E. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via nlp," in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 2016, pp. 196–205.

[4] S. Yang and H. Sahraoui, "Towards automatically extracting UML class diagrams from natural language specifications," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 396–403.

[5] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[8] M. A. Garzón, H. Aljamaan, and T. C. Lethbridge, "Umple: A framework for model driven development of object-oriented systems," in *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (saner)*. IEEE, 2015, pp. 494–498.

[9] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[10] W. Bian, O. Alam, and J. Kienzle, "Automated grading of class diagrams," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 700–709.

[11] M. B. Chaaben, L. Burgueño, and H. Sahraoui, "Towards using few-shot prompt learning for automating model completion," in *IEEE/ACM International Conference on Software Engineering (ICSE 2023 NIER track)*, 2023, in press.

[12] P. Singh, Y. Boubekeur, and G. Mussbacher, "Detecting mistakes in a domain model," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 257–266.

[13] Y. Boubekeur, G. Mussbacher, and S. McIntosh, "Automatic assessment of students' software models using a simple heuristic and machine learning," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–10.

[14] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[15] OpenAI, "GPT-4 technical report," 2023.

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

[17] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[18] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heinz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *arXiv preprint arXiv:2111.01243*, 2021.

[19] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[21] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[22] "GPT-3.5 models." [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[23] "Introducing chatgpt." [Online]. Available: https://openai.com/blog/chatgpt

[24] F. Wilcoxon, *Individual comparisons by ranking methods*. Springer, 1992.

[25] T. Shin, Y. Razeghi, R. L. L. IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, (EMNLP 2020)*. Association for Computational Linguistics, 2020, pp. 4222–4235.

[26] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," *arXiv preprint arXiv:2211.01910*, 2022.

[27] S. Diao, P. Wang, Y. Lin, and T. Zhang, "Active prompting with chain-of-thought for large language models," *arXiv preprint arXiv:2302.12246*, 2023.

[28] T. Lethbridge and R. Laganière, *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. McGraw Hill, 2004.

[29] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An NLP-based architecture for the autocompletion of partial domain models," in *Advanced Information Systems Engineering: 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28–July 2, 2021, Proceedings*. Springer, 2021, pp. 91–106.

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

[31] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram," *arXiv preprint arXiv:1211.0713*, 2012.

[32] B. Combemale, J. Gray, and B. Rumpe, "Chatgpt in software modeling," *Software and Systems Modeling*, vol. 22, 05 2023.

[33] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative ai in modeling tasks: An experience report with chatgpt and uml," *Softw. Syst. Model.*, vol. 22, no. 3, p. 781–793, may 2023. [Online]. Available: https://doi.org/10.1007/s10270-023-01105-5