Multi-step Iterative Automated Domain Modeling with Large Language Models

Yujing Yang
McGill University
Montreal, Quebec, Canada

Boqi Chen
McGill University
Montreal, Quebec, Canada

Kua Chen
McGill University
Montreal, Quebec, Canada

Gunter Mussbacher
McGill University
Montreal, Quebec, Canada

Dániel Varró *
Linköping University
Linköping, Sweden

ABSTRACT

Domain modeling, which represents the concepts and relationships in a problem domain, is an essential part of software engineering. As large language models (LLMs) have recently exhibited remarkable ability in language understanding and generation, many approaches are designed to automate domain modeling with LLMs. However, these approaches usually formulate all input information to the LLM in a single step. Our previous single-step approach resulted in many missing modeling elements and advanced patterns. This paper introduces a novel framework designed to enhance fully automated domain model generation. The proposed multi-step automated domain modeling approach extracts model elements (e.g., classes, attributes, and relationships) from problem descriptions. The approach includes instructions and human knowledge in each step and uses an iterative process to identify complex patterns, repeatedly extracting the pattern from various instances and then synthesizing these extractions into a summarized overview. Furthermore, the framework incorporates a self-reflection mechanism. This mechanism assesses each generated model element, offering self-feedback for necessary modifications or removals, and integrates the domain model with the generated self-feedback. The proposed approach is assessed in experiments, comparing it with a baseline single-step approach from our earlier work. Experiments demonstrate a significant improvement over our earlier work, with a 22.71% increase in the F_1-score for identifying classes, 75.18% for relationships, and a 10.39% improvement for identifying the player-role pattern, with comparable performance for attributes. Our approach, dataset, and evaluation provide valuable insight for future research in automated LLM-based domain modeling.

CCS CONCEPTS

• **Software and its engineering → Object oriented architectures**; **Object oriented architectures**; • **Computing methodologies → Machine learning approaches**.

KEYWORDS

domain modeling, large language models, few-shot learning, prompt engineering

ACM Reference Format:
Yujing Yang, Boqi Chen, Kua Chen, Gunter Mussbacher, and Dániel Varró . 2024. Multi-step Iterative Automated Domain Modeling with Large Language Models. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3652620.3687807

1 INTRODUCTION

Context: Domain modeling is a core process in model-driven software engineering (MDE), essential for building a system design from various sources of information, including documents and stakeholder interactions, etc. This process is typically performed manually by software engineers and domain experts, which can be time-consuming and highly dependent on human expertise. To mitigate this intensive manual effort, many approaches aim to automate this process [8, 16, 17, 21].

Recent advances in large language models (LLMs) have shown remarkable generalizability to tasks beyond natural language processing [23]. LLMs can perform various functions without supervised training on the specific task, using carefully designed input (called *prompts*). With different *prompt* designs, LLMs can achieve impressive performance on various tasks by only using a few labeled examples in the prompt. These advancements suggest significant potential for applying LLMs to complex tasks such as *fully automated* domain modeling.

Several approaches aim to automate this process using LLMs [4–6]. These methods typically consolidate all information into a single prompt, allowing the LLM to generate the domain model based on the given context. Our previous work [7] formulates the problem of automated domain modeling as a text-generation task, designing an approach to construct a single prompt and to create a domain model from the problem description. While these previous results highlight the potential of LLMs in modeling, two major problems of these approaches have also been identified: (1) the recall of identifying classes, attributes, and relationships is, in general, much

*Also with McGill University.

Partially supported the FRQNT-B2X project (file number: 319955), IT30340 Mitacs Accelerate, and the Wallenberg AI, Autonomous Systems and Software Program (WASP), Sweden.

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687807

lower than the precision and (2) no modeling patterns (that capture modeling best practices) have been identified in case of the evaluated domain models.

With the advance of research in LLMs, an increasing focus is put on integrating external or human knowledge into the prompt using a multi-step approach [10, 18, 22]. These approaches highlight that, for complex tasks, LLMs can benefit from breaking down the task into more simple sub-tasks and including human knowledge to solve these sub-tasks. However, this aspect has not been investigated so far in automated domain modeling.

Reflexion [18] adds self-reflective feedback as context for the LLM agent in the next generation. This self-feedback provides LLMs with a concrete direction to improve, helping them learn from prior mistakes and improve subsequent generations. Motivated by this approach, a self-feedback mechanism can be included in automated domain modeling to further enhance the quality of output models.

Objectives: This paper aims to improve the performance of *fully automated domain modeling* by including human knowledge and self-reflection. We propose a multi-step iterative automated domain modeling approach with LLMs to extract model elements and complex patterns from a textual-based problem description and construct a domain model. This paper also evaluates the proposed approach on an existing dataset and compares its performance to our previous single-step approach. Furthermore, this paper also strives to identify the advantages and limitations of the proposed approach for fully automated domain modeling.

Contributions: Given a textual domain description, we present a novel approach for fully automated domain modeling using LLMs. The specific contributions of this paper are the following:

- We propose a multi-step iterative automated domain modeling approach using LLMs combined with human knowledge, including an iterative process to identify best-practice patterns.
- Furthermore, our approach includes a self-reflection mechanism to generate internal self-feedback according to human knowledge and then integrate the generated self-feedback into the domain model for improvement.
- We carry out experiments with GPT-4 [14] to evaluate the proposed approach compared to our previous single-step approach. Specifically, this paper assesses the performance of generating domain models (including classes, attributes, and relationships) and the performance of identifying domain modeling patterns.

2 BACKGROUND

2.1 Problem Formulation

Domain modeling involves converting a textual description of the system in natural language into a structured domain model represented as a class diagram. Our previous work [7] treats domain modeling as a *text generation* problem, where a generative mapping $\mathtt{f}$ directly converts a domain specification d into a domain model M, i.e., $M = \mathtt{f}(d)$. In our enhanced multi-step approach, the mapping from input to output $\mathtt{f}$ is a multi-step LLM-based approach, and the output of $\mathtt{f}$ is a textual specification of M. Specifically, $\mathtt{f}$ is composed of a series of mappings from input to output $\mathtt{f} \equiv \mathtt{f}_n \circ \ldots \circ \mathtt{f}_2 \circ \mathtt{f}_1$, where each mapping $\mathtt{f}_i$ takes the output of the previous step(s) as input, namely, $M = \mathtt{f}(\mathtt{d}) \equiv \mathtt{f}_n(\ldots \mathtt{f}_2(\mathtt{f}_1(\mathtt{d})))$.

2.2 Patterns for Domain Model

The domain modeling design process has many common patterns that have been concluded from years of practice. These patterns can help improve the accuracy, modularity, and reusability of domain models. Commonly used patterns involve the *player-role pattern* [11] and *abstraction-occurrence pattern* [12].

In the player-role pattern, the player class (e.g., Person) may play different roles (e.g., Client and Resident) either concurrently or at distinct times. Moreover, since the role classes derive from an abstract role class (e.g., UserRole), this structure allows the abstract class to retain attributes common across roles while also enabling the differentiation of specific features within its subclasses.

This paper focuses on the player-role pattern because it is often used and LLMs perform poorly in identifying it in our prior work [7]. However, our approach can be generalized for other patterns.

2.3 Single-step Generator

A single-step LLM-based method [7] has been introduced for automating domain model generation. This method involves a *single* prompt, including a problem description, task description, and output format description, to generate the domain model with an LLM. The result demonstrates that the top-performing LLM, GPT-4, shows impressive domain understanding capability without explicit training. However, the single-step approach has limitations. Most critically, this approach often misses elements in the domain model (low recall). Additionally, the approach struggles to adhere to modeling best practices, such as the player-role pattern. Finally, it is constrained by the LLM's token limit, making it difficult to scale to large domain models.

2.4 Domain Model Evaluation

This paper follows the evaluation scheme and criteria of our previous work [7], which relies on reference domain models constructed by experts to evaluate the quality of a generated domain model.

2.4.1 Evaluation Scheme. Motivated by existing work [1], we categorize each model element into one of four categories: (1) *direct match* where the same element is also in the reference model, (2) *semantically equivalent* where the reference model contains an element equivalent to the current element, (3) *partial match* where the element achieves the functionality but can be further improved, and (4) *no match* where the element is not used in the reference.

The scoring function S can be defined for an element x as follows:

$$S(x) = \begin{cases} 1 & \text{if } x \text{ is in } \textit{exact match} \text{ or } \textit{semantically equivalent} \\ 0.5 & \text{if } x \text{ is in } \textit{partial match} \\ 0 & \text{if } x \text{ is in } \textit{no match} \end{cases} \quad (1)$$

The scoring function will then be used to evaluate the quality of an auto-generated domain model.

Evaluation Criteria. We evaluate the quality of the output domain model by using the standard statistical metrics of precision, recall, and F_1-score over the classes, relationships, and attributes.

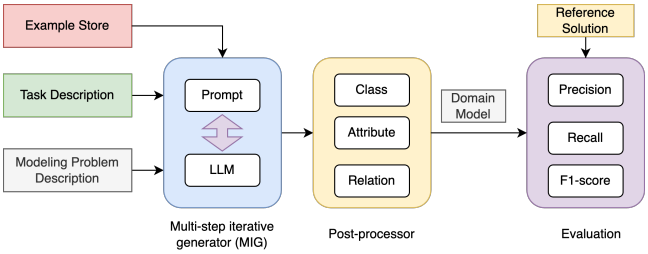

Figure 1: Architecture of multi-step automated domain modeling approach

Precision measures the overall correctness of generated model elements. For example, let C be the set of all classes and enumerations in the generated model of size $m = |C|$, the precision of classes in generated models can be expressed as

$$Precision_C = \frac{\sum_{i=1}^{m} S(C_i)}{m}, \tag{2}$$

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 C_{gt} be the set of classes and enumerations in the *reference model* (ground truth) of size $n = |C_{gt}|$, the recall of classes can be expressed as

$$Recall_C = \frac{\sum_{i=1}^{n} S(C_i)}{n}. \tag{3}$$

Finally, we use the classical F_1-score definition:

$$F_{1C} = \frac{2 \times Precision_C \times Recall_C}{Precision_C + Recall_C}. \tag{4}$$

Metrics for attributes or relationships can be computed by substituting C with the set of attributes or relationships, respectively.

3 APPROACH

3.1 Architecture

Figure 1 illustrates the architecture of the proposed multi-step automated domain modeling approach. To generate a domain model, we provide a *task description* to describe the overall domain model generation task, a text-based *modeling problem description* written in natural language to describe a problem domain, and an *example store* that contains few-shot examples explaining the definition of model elements and patterns.

Next, the **multi-step iterative generator (MIG)** takes the input information, formulates prompts for an LLM iteratively, and outputs a textual-based domain model. The designation *multi-step* describes the framework of the generator. Instead of generating a complete domain model in a single step, the tasks are divided into sequential subtasks while each task outputs an incomplete *partial model*. For each subtask, a combination of inputs is used to formulate the prompt, including a subtask description, a modeling problem description, the output from the previous step(s), and an output format description. A sequence of these outputs can be combined into the final complete model. Throughout the multi-step process, the generator continually revises the partial model based on the self-feedback from previous steps, allowing for the gradual accumulation of complex patterns and reducing the likelihood of errors or omissions in the final model. The term, *iterative*, further highlights that the approach refines and builds upon the output through repeated cycles, which is especially significant in identifying domain modeling patterns (e.g., the player-role pattern) within the model.

Finally, a **post-processor** is employed to refine the LLM's output by systematically extracting key elements such as classes, attributes, and relationships. This post-processor utilizes rule-based algorithms to carefully analyze the LLM-generated text, identifying and isolating the relevant components. It then organizes these elements into a structured and accurate final domain model, ensuring that all critical aspects are correctly captured and aligned with the expected domain model structure.

Our experimental **evaluation** involves comparing the generated domain model to a reference solution created by a human modeling expert using standard statistical metrics (see Section 2.4.1). We use these metrics to evaluate each type of model element separately, including classes, attributes, and relationships.

3.2 Multi-step Iterative Generator (MIG)

As fully automating domain modeling is a complex and challenging task, we mimic how MDE engineers solve domain modeling tasks and split the single-step process into smaller, pre-defined subprocesses. By breaking down the domain modeling process into distinct phases, MIG only tackles a subtask within each step. An LLM is involved in each step.

For each pre-defined subtask, MIG combines textual inputs, including the output from previous steps, formulates the information as prompts, and sends it to an LLM. The output is then used as the input for the following step. MIG contains four sub-processes as shown in Figure 2: (1) identify classes and attributes, (2) identify the player-role pattern, (3) self-reflection, and (4) identify relationships. By breaking the single step into multiple processes and identifying the player-role pattern iteratively, human instructions and examples can be included in the individual process to improve the performance of identifying modeling elements. Furthermore, the self-reflection process provides internal self-feedback from pre-defined instructions to improve itself without any external feedback.

The detailed workflow is shown in Figure 2, including the input and processes. We use the H2S domain from an existing work [7] as the running example to illustrate the output for each step:

"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. [...] 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, and 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

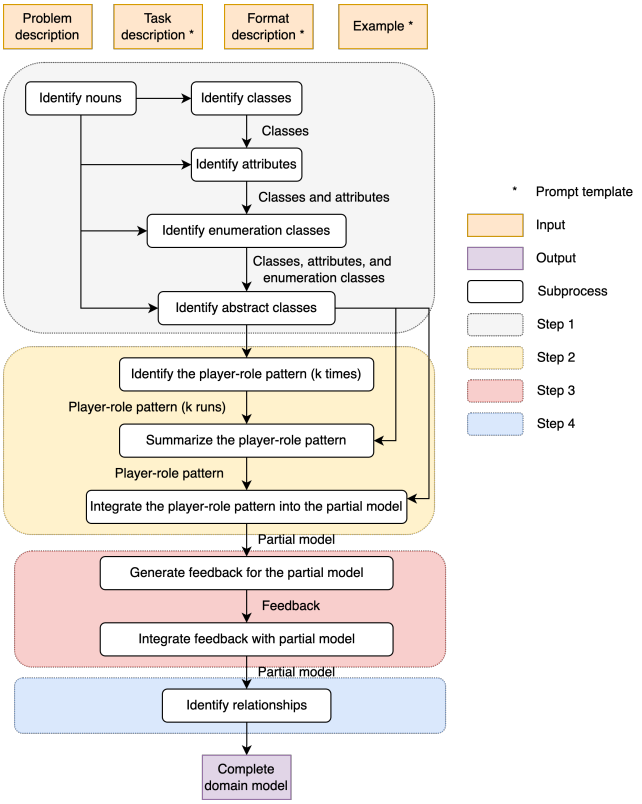

Figure 2: Detailed flowchart of MIG

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. [...]"

3.2.1 Step 1. Identify classes and attributes.

Identify nouns. We perform noun analysis on the problem description to identify potential candidates for class names, attribute names, role names for associations, and literals for enumeration classes. For example, identified nouns for H2S with the desired output format are:

```
Helping Hand Store, resident, second hand article, ...
```

Identify classes. The approach identifies classes using the problem description and the identified nouns by prompting an LLM. Sample identified classes in the H2S domain include:

```
HelpingHandStore, SecondHandArticle, ...
```

Identify attributes. Attributes use only primitive data types, for example, *String*, *Integer*, *Date*, *Time*, and *Boolean*. The prompt instructs the LLM to avoid modeling more complex data as classes. The identified classes with attributes following the example are:

```
HelpingHandStore(), SecondHandArticle(string codeRFID,
    boolean discarded, String category)
```

Identify enumeration classes. In this step, problem description and identified nouns, classes, and attributes from previous steps are evaluated to decide whether they should be modeled as enumerations instead. An example enumeration class continuing with the example is shown below:

```
enum ItemCategory()
Client(ItemCategory neededCategories)
```

Identify abstract classes. Our approach identifies abstract classes (with the "abstract" keyword in the output) by using the problem description, nouns, and the revised classes. An example abstract class with the desired output format is shown below:

```
abstract Item(string description, string dimension,
    int weight, Date requestedPickedDate)
```

3.2.2 Step 2. Identify the Player-Role pattern. Identifying the player-role pattern poses challenges as our single-step automated domain modeling approach with LLMs failed to accurately detect this pattern [7]. To enhance detection accuracy, we implement several strategies: (1) We establish a separate process dedicated to detecting the player-role pattern. Once identified, the pattern is integrated into the partial model from Step 1. And (2) we apply N-shot learning in the prompt design, with five classic examples of the pattern, each accompanied by a brief overview of the relevant classes and the solution in the expected format. (3) We conduct multiple assessments (k times) of the player-role pattern for each model. Each assessment follows identical guidelines and information, but it is applied with a non-zero temperature value to introduce variability. Then, the k assessments are combined with the output for the player-role pattern and are integrated into the partial model.

Identify the player-role pattern. To accurately identify the player-role pattern, we incorporate both the problem description and the initial output, which includes nouns, classes (possibly abstract), attributes, and enumerations as inputs. The provided instructions explain the player-role pattern, offer examples, and outline the expected output format. We specify two key terms: *abstract* indicates whether a class is abstract, and *inherit* denotes the relationship between a subclass and its superclass because they are essential components in the player-role pattern.

We adopt N-shot learning in our prompt design, where N examples are provided to the LLM as input-output pairs to learn the desired output's format and content. Each example consists of two parts: a brief description of a problem within a specific domain and a textual representation of a reference player-role pattern for that example. The N-shot examples can help the model identify class names, attribute names, and the relationships between superclasses and subclasses. We present five classic examples of player-role patterns to illustrate this process. One example of the output from this step is shown below (detailed attributes are ignored for space):

```
Person(...),
abstract UserRole(),
Resident(...) inherit UserRole,
Volunteer(...) inherit UserRole,
Client(...) inherit UserRole
```

Summarize the player-role pattern. This step of the approach focuses on summarizing the player-role pattern based on prior generations of the player-role pattern. The inputs to this process

include a detailed problem description, a compilation of outcomes from previously generated solutions for the player-role pattern, and an existing partial model. In scenarios where the provided information does not suggest a discernible player-role pattern, the process is equipped to explicitly indicate the absence of such a pattern, outputting "No player-role pattern identified". The summarized player-role pattern has the same format as the previous step. The following instructions are included in the prompt:

```
Identify the player-role pattern from the description
provided regarding the five result lists. Output the
mostly like player-role pattern according to the 5
results you have. You do not need to include
everything from the 5 results you have; only include
the classes you think are correct. Combine the 5
results you have and make the final solution that
makes sense to you. Do not output other classes that
are not included in the player-role pattern. If there
is not any player-role pattern, simply say "No
player-role pattern identified".
```

Integrate the player-role pattern into the partial model. This step integrates the player-role pattern into the previous partial model from Step 1 with the problem description and the extracted patterns. This step involves a detailed evaluation to ensure that the inclusion of the player-role pattern is both necessary and beneficial to the domain system. The primary instruction is to evaluate the player-role pattern based on its necessity and to evaluate the model elements in the partial model concerning the elements in the pattern. The following instructions are included in the prompt:

```
1.Analyze the generated classes to see if they are
needed. Some generated classes may not be at the
right level of abstraction. Drop the classes if
they are not necessary to describe the system.
2. Evaluate the player-role patterns to see if they
are necessary. Not all systems need the player-
role pattern. Since player-role patterns can be
complex in implementation, only use them if it is
necessary. If the abstract classes and their
subclasses are necessary, do not use a player-
role pattern.
3. Combine the two versions and make a solution that
is consistent with both versions. Do not have
duplicate classes in the final solution.
```

3.2.3 Step 3. Self-reflection. During the self-reflection process, the LLM is used to improve the output of itself. Specifically, it includes two steps: (1) generate self-feedback for the partial domain model, and (2) improve the partial domain model according to the self-feedback.

Generate self-feedback for the partial model. Similar to the framework of self-refinement [13] and reflexion [18], generating useful reflective self-feedback is challenging since it requires a good understanding of where the model made mistakes as well as the ability to generate a summary containing actionable insights for improvement. We analyzed the domain model generated from prior experiments, which revealed two main issues: (1) many nouns extracted from the problem description in the initial steps are misidentified as classes, and (2) some generated generalization relationships are misleading.

To generate self-feedback to solve the problems above, the LLM evaluates each class with its attributes separately. The main objective is to evaluate if the class and attribute are at the right level of abstraction. The evaluation and self-feedback generation processes are guided by two instructions corresponding to the two main issues:

1: Classes that are overly detailed or contain minimal attributes (none or one) should be modeled differently. For example, if a class does not have meaningful attributes, its attributes should be reassigned to a more suitable class, and the original class shall be removed.
2: Some generalization relations from the *player-role pattern* may still be unnecessary. These relations should only be used when there is a clear relation between the superclass and subclasses to reduce redundancy and enhance clarity. Some subclasses may be better modeled as instances of other classes and should be removed.

To generate self-feedback for the current partial model, the input prompt includes a problem description, a partial model revised from previous steps, and the following instructions:

```
Given the class list for the problem description,
write a comment for each class with its attributes.
Evaluate if it is at the correct level of abstraction
to be included in the software system. Many classes
may not be needed or necessary, for example
- If class A is too detailed to be included in the
system, consider removing it.
- If class A does not contain any attributes or only
contains 1 attribute, consider moving the attribute
of class A to another class and removing class A.
- For the enumeration class, evaluate if it should
be captured by an attribute and if its literals are
necessary.
- For the subclasses, evaluate if they are necessary
to be present in the system.
You can write general comments for each class, and
evaluate if the class is necessary. If not, provide
a solution to change it.
```

One example of the generated self-feedback for the class Dimension (String length) is shown below:

```
It is unnecessary as the dimension can be an
attribute of the Item class.
```

Integrate self-feedback into the partial model. Next, we use the generated self-feedback to revise and improve the partial model. To integrate the self-feedback with the previous model, we use the problem description, the partial model, and self-feedback for each class as input.

For example, given the self-feedback shown earlier, the Dimension is removed, and the attribute String dimension is added to the Item class.

3.2.4 Step 4. Identify Relationships. With the completed classes and attributes, we identify relationships between the classes. In this step, we aim to identify generalization, composition, and associations.

Composition. We use the keyword *contain* to indicate that a composite class contains a part class. Multiple compositions are allowed within a domain model, generally with a multiplicity from one of the following options [0..*, 1, 0..1, 1..*]. An example of composition is shown below:

```
1 H2S contain * Person
```

Generalization. The keyword *inherit* is used to indicate that a subclass inherits properties (e.g., attributes and relationships) from a superclass. A domain model may contain multiple generalizations. Use cases of a generalization relationship include player-role patterns. An example of generalization is shown below:

```
Resident inherit UserRole
```

Association. An association defines all other relationships between two classes or with the same class. The keyword *associate* is used to indicate that a class associates with another class. An example is shown below:

```
1 Resident associate * Item
```

The complete set of prompts can be accessed in the repository[1].

4 EXPERIMENTAL EVALUATION

4.1 Overview

This chapter aims to assess the capability of our approach: the multi-step iterative LLM-based automated domain modeling system. We use the same experiment setting to evaluate our approach against the baseline approach: our previous single-step approach we introduced in Section 2. More concretely, we aim to investigate the following two research questions (RQs):

- RQ1: What is the performance of the MIG approach compared to our previous single-step approach?
- RQ2: How effectively are player-role patterns identified with our MIG approach?

4.2 Experimental Settings

Evaluated LLMs. As our approach is based on generative LLMs, choosing an LLM for the experiment is crucial. In our MIG approach, each step accesses one LLM. To assess the ability of our approach comprehensively, we apply the same LLM in each step. `GPT-4` is one of the latest models released by OpenAI [15]. `GPT-4` can solve complex tasks with higher performance than any of the previous OpenAI models with broader general knowledge and more advanced reasoning capabilities. We use GPT-4 in the evaluation to assess the limit on how LLMs can generate domain models with our approach. Nevertheless, our approach can be easily adapted to other LLMs.

[1]https://github.com/YujingYang666777/DomainModelGeneration

Temperature Value. In our experiment, we apply a temperature value of 0.7 in the process of identifying the player-role pattern. For other steps in our approach and in the baseline approach, we apply a temperature value of 0 to reduce the nondeterminism in the generation.

Test Set. We use a test set of eight domain models (Table 1) from previous work [7] to evaluate our proposed approach compared to the baseline approach. The examples are taken from projects or exams from an undergraduate-level, model-driven programming course and have been used across multiple course offerings, covering a wide range of domains. Each domain model consists of a description of the domain system and an associated ground truth domain model developed by modeling experts. Each example represents a software system, and three of the domain models require the use of player-role patterns. A high-quality reference solution (model) is available for each example.

Evaluation Procedure. Due to the limitation of automated evaluators, we choose to manually evaluate the generated domain models to precisely measure the performance of our approach. To ensure consistency and fairness in grading, only one author grades all generated domain models. Throughout this process, any issues encountered by the grader are discussed and resolved together with other domain experts. The code and evaluation test set can be accessed in the repository.

4.3 RQ1: Quality of Generated Models

4.3.1 Rationale. RQ1 aims to evaluate the performance of our MIG approach compared to the baseline single-step approach.

In this research question, we aim to compare the performance of the baseline and our MIG approach. This shows if our approach has improved compared to the baseline and how far our approach is from the human-expert performance score for this task.

Furthermore, we carry out a more fine-grained analysis by highlighting which modeling aspects the LLMs struggle with. Particularly, we compare the performances of the LLMs when recovering classes, attributes, and relationships.

4.3.2 Result. The average result of our previous single-step and our proposed MIG approach are presented in Table 2. The bold text indicates an improvement in the performance.

Class, attribute, and relationship. For both approaches, the performance of identifying classes is higher than the one of attributes and higher than the one of relationships. One reason is that a relationship is more complex, requiring the correct classes on both sides and two correct multiplicities.

Precision, recall, and F_1-score. The precision for classes and attributes is marginally reduced in our MIG approach by 5.45% (from 0.8483 to 0.8021) and 14.24% (from 0.5626 to 0.4825), respectively. Conversely, precision for relationship identification increases. This suggests that a comparable proportion of elements within our generated model are correctly identified and aligned with the solution. For our previous single-step approach, the precision is higher than the recall for almost all model elements, indicating that the identified elements are accurate, but there are many missing elements.

Table 1: Collected modeling examples and their domains

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

Table 2: Comparison of precision, recall, and F_1-score for our previous single-step and proposed MIG approaches for each model element type

	Single-step Approach			MIG Approach		
Model Element	Precision	Recall	F_1-score	Precision	Recall	F_1-score
Class	**0.8483**	0.5003	0.6280	0.8021	**0.7502**	**0.7706**
Attribute	**0.5626**	0.5329	**0.5403**	0.4825	**0.5732**	0.5176
Relationship	0.2867	0.1420	0.1781	**0.3256**	**0.3027**	**0.3120**

In our proposed MIG approach, the recall is increased for all three elements, especially for classes and relationships. Specifically, the MIG approach improves the recall of classes by 50.03% (from 0.5003 to 0.7502) and improves the recall of relationships by 113.17% (from 0.1420 to 0.3027). This indicates that our generated model covers many more elements in the solution model.

The proposed MIG approach has similar precision to our previous single-step approach while improving the recall significantly, resulting in increased F_1-scores. Specifically, our proposed approach improves the F_1-score of identifying classes by 22.71% (from 0.6280 to 0.7706) and improves the F_1-score of identifying relationships by 75.18% (from 0.1781 to 0.3120). The F_1-score is similar for attributes. The improvements in recall and F_1-score come with a trade-off in precision, particularly for attributes, where precision dropped by 14.24% (from 0.5626 to 0.4825). This trade-off suggests that while the MIG approach successfully identifies a broader range of elements, it may also introduce some incorrect elements, especially in more complex or ambiguous areas like attribute identification.

While the results obtained are promising, there is still room for improvement, especially for attributes and relationships. One potential future direction is to integrate rules with LLMs in some steps. Another direction is to improve components within our MIG approach, for example, by adding more self-reflection processes.

Answer to RQ1. This RQ shows that our proposed approach overall improves the performance of automating domain modeling compared to our previous single-step approach. With a slight decrease in attribute performance, our approach significantly increases the performance of classes and relationships, especially in terms of recall.

4.4 RQ2: Performance of Identifying Player-role Patterns

4.4.1 Rationale. RQ2 aims to evaluate the performance of identifying advanced patterns for our MIG approach. In our proposed approach, we design a step in the multi-step iterative generator to identify the player-role pattern explicitly. In the research question, we further investigate the performance of identifying the player-role pattern and compare it with our previous single-step approach.

Table 3: Precision, recall, and F_1-score of three approaches of identifying player-role patterns: zero-shot single-step, two-shot single-step, and multi-step

Approach	Precision	Recall	F_1-score
Zero-shot Single-step	0.9242	0.6143	0.7380
Two-shot Single-step	**1**	0.6571	0.7931
Multi-step	0.83	**0.8**	**0.8147**

For evaluation, we compare the performance of the three approaches, zero-shot single-step, two-shot single-step, and multi-step, using precision, recall, and F_1-scores for the identified player-role patterns.

Single-step Approach. In our previous single-step approach as introduced in Section 2, we formulate all input information into one prompt. To investigate the effect of adding relevant examples of the player-role pattern, two prompting techniques are compared, **zero-shot prompting** and **two-shot prompting**. The zero-shot prompt only provides a high-level description, while the two-shot approach also provides two-generation examples.

MIG Approach. Our proposed MIG approach includes the step to identify the player-role pattern for k times, summarizing the result of the identified player-role pattern during k times, and integrating it into the partial model. During our experiment, we set **k=5**, which means we apply the same prompt to identify the player-role pattern five times. We also set the **temperature value** to **0.7** in the process of identifying the player-role pattern and set it to **0** for the other steps to reduce the nondeterminism.

4.4.2 Result. Table 3 shows the precision, recall, and F_1-score for zero-shot and two-shot prompting approaches. Two-shot prompting has higher performance in all three metrics compared with zero-shot prompting. This shows that within a single step, adding examples improves the performance of identifying a player-role pattern.

As shown in Table 3, in comparison with the zero-shot single-step approach, our MIG approach has a slightly lower precision (decrease by 10.19%, from 0.9242 to 0.83) but a much higher recall (increase by 30.23%, from 0.6143 to 0.8), resulting in a higher F_1-score (increase by 10.39%, from 0.7380 to 0.8147). The decrease in precision may be because there are many more classes and attributes identified with the MIG approach, resulting in more potentially unnecessary elements.

In comparison with the two-shot single-step approach, our MIG approach also achieves a higher recall and a slightly higher F_1-score (increase by 2.72%, from 0.7931 to 0.8147). This comparison further demonstrates the effectiveness of our proposed approach in extracting the complete player-role pattern.

For potential improvements, one of the future directions is that we can design components to identify other advanced patterns, for example, the abstraction-occurrence pattern.

Answer to RQ2. Overall, our proposed MIG approach has a better performance compared to our previous single-step approach, with a higher recall and a slightly higher F_1 score.

4.5 Threats to Validity

Internal Validity. The output of an LLM can be slightly different for each run. To address this variation, we apply a temperature of 0 to reduce the nondeterminism for most steps except for detecting the player-role pattern, where we intentionally introduce variance. While this does not guarantee identical outputs each time, it greatly reduces randomness, making the results more stable and predictable. Besides, we also keep all generated models for future research. Furthermore, we address this variation by experimenting with a set of eight diverse domain examples. The evaluation is a manual process done by one author, which may introduce bias. We mitigate this bias by following the grading scheme from previous work [7] Besides, we also have multiple rounds of evaluation and discuss the results with other domain experts.

External Validity. Due to the lack of benchmark data sets for domain modeling with both problem description and reference solution, we adapt eight modeling examples with reference models created by experts from previous work [7]. Furthermore, models in this dataset are from undergraduate modeling courses i.e., representative of domain modeling exercises in education scenarios. Our proposed approach 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 [8, 16, 17, 21].

5 RELATED WORK

Techniques and frameworks for large language models. With the advancement of large language models (LLMs), various prompting techniques have been developed to enhance their ability to address complex real-world problems. Few-shot prompting, proposed by Brown et al. [2], introduces a paradigm where the model is provided with a limited number of task demonstrations during inference but does not need fine-tuning or weight updates. Chain-of-thought prompting, introduced by Wei et al. [19], allows models to decompose multi-step problems into intermediate steps for complex tasks and allocates additional computation to problems that require more reasoning steps. Building on research in prompting techniques that enhance LLM reasoning, Yao et al. [22] proposed ReAct, which integrates reasoning, action, and decision-making in LLMs for complex tasks. This interactive approach allows the model to create and adapt plans dynamically (reason to act) and interact with external environments like Wikipedia to integrate information (act to reason). LangChain [10], an open-source Python library, helps developers build LLM applications with features like prompt templates, customizable agents, retrieval modules, memory, and callback functions.

This paper introduces a multi-step iterative approach to fully automating domain modeling with LLMs by breaking the entire task into more manageable subtasks. By providing focused examples within each segment, our framework enhances the learning process by enhancing LLMs' reasoning and self-reflection ability, as well as their knowledge of domain modeling.

Large language models for MDE. Model-driven Engineering (MDE) is a widely used software development methodology, particularly valuable for developing large-scale systems. With the advancement of LLMs, various LLMs have been integrated into MDE, significantly improving the development process. Chaaben et al. [5] propose using GPT-3 for model completion by generating related elements for a partial model. They create examples using classes and their relationships, then form prompts with few-shot learning. However, their approach lacks a domain problem specification and relies heavily on the partial model. It also does not include attributes, multiplicity, types of classes, or relationships. Cámara et al. [4] present the use of ChatGPT to build UML class diagrams enriched with OCL constraints in an interactive mode. Their findings show that, in contrast to code generation and completion, the performance of ChatGPT for software modeling is still quite limited. In our previous publication [6], we report on early experimental results regarding the potential use of GPT-4 to develop goal-oriented models. While it is valuable to include syntax information for creating a goal model in the prompt, the amount of domain information has a limited effect on the responses of GPT-4. Many elements generated by GPT-4 may be either incorrect or rather generic and hence not very conducive to highlight important conflicts among stakeholders in the domain.

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. Furthermore, our multi-step approach decomposes the process into several steps to include more examples and human instructions and iteratively identifies advanced patterns from the description.

Automated Domain Modeling. Recently, LLMs have also been used for automated domain modeling [7]. With LLMs, these approaches are capable of generating a domain model from text description without any human involvement. However, these approaches generate the entire domain model in a single step, limiting both their performance and scalability. In this paper, we discuss a multi-step iterative approach using LLMs.

Many existing research efforts on automated domain modeling utilize rule-based linguistic processing methods or statistical natural language processing methods to directly derive complete domain modeling solutions like UML class diagrams [16, 17] or provide modeling assistance and suggestions [3, 20] from textual descriptions in natural language.

Rule-based methods rely on predefined rules and heuristics to model the knowledge and structure of a specific domain. These

rules include hand-written grammatical templates and heuristics in linguistics. An example of a rule-based method presents an algorithm with 23 heuristics to identify model elements from user stories [16] automatically. The designed system processes user stories written in a specific format and generates a conceptual model as output. Another example is proposed by Herchi et al. [9]. The system uses an NLP toolkit to decompose the user need in natural language and then uses linguistic rules (e.g., *All nouns are converted to entity types*) to extract UML concepts. Statistical methods emphasize using natural language processing techniques to extract domain models from requirement documents. Burgueño et al. [3] designed a framework to suggest new domain model elements for a given partially completed model, including classes, attributes, and relationships, using two word-embedding models. Saini et al. [17] introduced a novel methodology for extracting domain concepts and relationships from problem description, generating and processing decision points, resulting in multiple possible configurations for domain models.

6 CONCLUSION

This paper proposes a multi-step iterative LLM-based automated domain modeling approach, which extracts model elements from a textual-based problem description and constructs the domain model, *without* any human interaction *or* supervised training. Our proposed approach is built upon our previous single-step approach but splits the task of domain model creation into subtasks based on how humans tackle domain model generation tasks. By solving each subtask in a separate step, our approach is able to incorporate more instructions and examples in each step, and a self-reflection mechanism. Besides, our approach includes an explicit step to identify advanced patterns, such as the player-role pattern, and to generate and integrate self-feedback without any external interaction. Our proposed approach significantly increases the recall compared to our previous single-step approach, while maintaining similar precision, which indicates that our proposed approach can cover more model elements in the solution model.

For future research, our proposed approach can be further extended to identify other advanced model elements and patterns. Besides, our approach can be enhanced with a rule-based component to check the generated model, including the format and analysis of each modeling element, and create self-feedback to improve the output. Additionally, we could explore the inclusion of a human-in-the-loop process, allowing domain experts to contribute their knowledge or guide the generative AI more effectively at each step. We could also explore other generative AI instead of GPTs and compare their performances. Furthermore, our evaluation used a set of eight domain models, but further analysis could involve comparing the results across different models to understand why some models performed better or worse. Additionally, while our evaluation has focused on comparing our previous single-step and proposed multi-step approaches, it would be valuable to conduct further comparisons with pre-existing domain modeling techniques that were used before LLMs became prevalent.

REFERENCES

[1] Weiyi Bian, Omar Alam, and Jörg Kienzle. 2019. Automated Grading of Class Diagrams. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 700–709. https://doi.org/10.1109/MODELS-C.2019.00106

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Loli Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, and Jordi Cabot. 2021. 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, 91–106.

[4] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the Assessment of Generative AI in Modeling Tasks: An Experience Report with ChatGPT and UML. *Softw. Syst. Model.* 22, 3 (May 2023), 781–793. https://doi.org/10.1007/s10270-023-01105-5

[5] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. 2023. Towards using few-shot prompt learning for automating model completion. In *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 7–12.

[6] Boqi Chen, Kua Chen, Shabnam Hassani, Yujing Yang, Daniel Amyot, Lysanne Lessard, Gunter Mussbacher, Mehrdad Sabetzadeh, and Dániel Varró. 2023. On the use of GPT-4 for creating goal models: an exploratory study. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 262–271.

[7] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Mussbacher, and Dániel Varró. 2023. Automated Domain Modeling with Large Language Models: A Comparative Study. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 162–172.

[8] Jan Franců and Petr Hnětynka. 2011. 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, 34–47.

[9] Hatem Herchi and Wahiba Ben Abdessalem. 2012. From user requirements to UML class diagram. *arXiv preprint arXiv:1211.0713* (2012).

[10] LangChain-AI. 2023. LangChain. https://python.langchain.com/docs/get_started/introduction. Accessed: 2023-02-20.

[11] Craig Larman. 2012. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India.

[12] Timothy Christian Lethbridge and Robert Laganiere. 2005. *Object-oriented software engineering*. Vol. 11. McGraw-Hill New York.

[13] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651* (2023).

[14] OpenAI. 2023. GPT-4 Technical Report. https://arxiv.org/pdf/2303.08774.pdf. arXiv:2303.08774 [cs.CL]

[15] OpenAI. 2024. gpt-4-and-gpt-4-turbo. https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo.

[16] Marcel Robeer, Garm Lucassen, Jan Martijn E. M. van der Werf, Fabiano Dalpiaz, and Sjaak Brinkkemper. 2016. Automated Extraction of Conceptual Models from User Stories via NLP. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 196–205. https://doi.org/10.1109/RE.2016.40

[17] Rijul Saini, Gunter Mussbacher, Jin L. C. Guo, and Jörg Kienzle. 2022. Machine Learning-Based Incremental Learning in Interactive Domain Modelling. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems* (Montreal, Quebec, Canada). ACM, 176–186.

[18] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366* (June 2023).

[19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[20] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. 2022. Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling* 21, 3 (2022), 1071–1089.

[21] Song Yang and Houari Sahraoui. 2022. 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*. 396–403.

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[23] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223* (2023).