# Application of the Tree-of-Thoughts Framework to LLM-Enabled Domain Modeling

Jonathan Silva[1(✉)], Qin Ma[1], Jordi Cabot[1,2], Pierre Kelsen[1], and Henderik A. Proper[3]

[1] University of Luxembourg, Esch-sur-Alzette, Luxembourg
{jonathan.silva,qin.ma,jordi.cabot,pierre.kelsen}@uni.lu
[2] Luxembourg Institute of Science and Technology, Esch-sur-Alzette, Luxembourg
jordi.cabot@list.lu
[3] TU Wien, Vienna, Austria
henderik.proper@tuwien.ac.at

**Abstract.** Domain modeling is typically an iterative process where modeling experts interact with domain experts to complete and refine the model. Recently, we have seen several attempts to assist, or even replace, the modeler with a Large Language Model (LLM). Several LLM prompting strategies have been attempted, but with limited success. In this paper, we advocate for the adoption of a Tree-of-Thoughts (ToT) strategy to overcome the limitations of current approaches based on simpler prompting strategies. With a ToT strategy, we can decompose the modeling process into several sub-steps using for each step a specialized set of generators and evaluators prompts to optimize the quality of the LLM output. As part of our adaptation, we provide a Domain-Specific Language (DSL) to facilitate the formalization of the ToT process for domain modeling. Our approach is implemented as part of an open source tool available on GitHub.

**Keywords:** Domain Modeling · Large Language Models · Tree of thoughts

## 1 Introduction

Domain modeling is the process of creating conceptual models of a specific domain to serve a particular purpose. This process typically involves a collaborative effort [12] between domain experts, who provide insights and knowledge about the domain, and modeling experts, who are skilled in abstracting the relevant aspects of the domain and expressing such an abstraction using a modeling notation, which can be either textual or graphical. Executing the domain modeling process is cognitively demanding due to many factors [16]. This includes the inherent complexity of the problem domain, the involvement and coordination of various stakeholders, and the need for the modeling abstraction to be constrained


W. Maass et al. (Eds.): ER 2024, LNCS 15238, pp. 94–111, 2025.
https://doi.org/10.1007/978-3-031-75872-0_6

by the modeling purpose [13]. Further evidence of this cognitive demand is provided by Feltus et al. [10], where they suggested that modeling experts should simultaneously manage at least four conceptions in their mind to successfully create a domain model. These are: (1) a comprehensive understanding of the domain, (2) a clear conception of the modeling purpose, (3) an abstraction of the domain that aligns with the modeling purpose, and (4) a conception of the domain model as an artifact that represents the abstracted domain in terms of a modeling language.

Manifold modeling languages exist, including both General-Purpose Modeling Language (GPML) and Domain-Specific Modeling Languages (DSML), to capture different aspects of a domain [2]. For example, in the context of Model-Driven Software Engineering (MDSE), an Entity Relationship (ER) model [7] or an UML class diagram [1] can be used to capture the static data and information structure of the system, while an UML sequence diagram to illustrate the dynamic behavior of the system. To enable structural modeling of systems, both ER and UML class diagram provide a similar set of key modeling elements. These include entity types (or classes in UML terminology), inheritance relation between entity types, and relationship types (equivalent of UML associations). Furthermore, entity types (classes) can have attributes (properties), and both attributes and relationships may be constrained by cardinality specifications. Beyond ER, UML class diagram also supports more sophisticated modeling elements as first-class citizens, such as defining containment as a special kind of association to represent the whole-part relationships, and the use of association classes to further specify properties uniquely associated to relationships.

Research to facilitate the domain modeling process and consequently alleviate the cognitive demand associated to it has been proposed in different approaches, focusing on different modeling activities during the process, such as creating domain models from textual requirements, completing partially created models, and repairing inconsistencies in models [2]. For example, an approach to support human modelers is to implement Natural Language Processing (NLP) techniques to process textual requirements. These techniques have existed for a long time, and recent research has used heuristic rules to identify model elements [3], and Machine Learning models to classify them [14].

The recent advancements in generative AI such as Large Language Models (LLMs) has triggered a new trend in research towards supporting domain modeling, by leveraging the language processing capabilities of LLMs to create domain models. Both the domain description and the modeling purpose, articulated in natural language, are provided as input to an LLM, referred to as "prompts". The LLM then generates outputs, also in natural language, which define the elements of the domain models in a specified notation [11]. Among others, two factors play a crucial role in the success of LLMs for the domain modeling task: the inherent trained capability of the LLM and the choice of prompting techniques, i.e., how the prompts are designed. A range of prompting techniques have been tested with two LLMs: GPT-3.5 and GPT-4 [5,6,8,11], spanning from the simplest input-output (IO) prompting (aka. zero-shot prompting), to IO prompting

enhanced with in-context learning [4] (aka. few-shot prompting), and finally to Chain-of-Thought (CoT) prompting [15]. Despite their promising performance in recognizing classes and relationships, these prompting techniques all have shown limitations in (1) correctly classifying relationships [6], (2) identifying complex modeling constructs such as association classes [8], and (3) leveraging best modeling practices (i.e., applying design patterns) [6].

Meanwhile, a new prompting technique: the Tree-of-Thoughts (ToT) framework, has been proposed [17]. This technique guides LLMs to solve a problem by explicitly decomposing it into a series of intermediate steps and considering multiple reasoning paths at each step, thus forming a tree of thoughts. LLMs are then instructed to explore the tree and heuristically evaluate intermediate thoughts towards the final solution. The ToT framework outperforms all existing prompting techniques in solving complex reasoning-based problems, a category to which our domain modeling problem belongs.

In this paper, we investigate the potential of this new prompting technique for domain modeling. More specifically:

- We propose a way to decompose domain modeling tasks using UML class diagram modeling as an example;
- We recommend our choice of the strategies for generator prompts and the evaluator prompts, as well as the search algorithm, among the options offered by the ToT framework;
- We develop a domain-specific language (DSL) to facilitate the configuration of the ToT framework for domain modeling tasks, and to automate their execution with LLMs;
- We validate our work by conducting experiments with GPT-4.

The rest of the paper is organized as follows. In Sect. 2 we review related work. Then, in Sect. 3, we review the ToT framework and design considerations. Next, in Sect. 4, we will present our approach to designing the ToT with domain modeling tasks. In Sect. 5 we propose the implementation of a DSL to configure the ToT framework for domain modeling. Then, in Sect. 6, we present experiments to validate our implementation. Section 7 reports on the experiments conducted using the ToT framework for domain modeling. Before concluding, Sect. 8 provides information about the support to our DSL.

## 2 SOTA: LLM-Enabled Domain Modeling

The LLMs have the ability to recognize the task requested by the context provided in the prompt, this is known as In-context learning [4]. The IO prompting technique uses this ability combining task instructions and examples. The zero-shot prompting is an IO prompting that relies on the ability developed by the LLM during the training to recognize the task [4]. The performance of LLMs can improve using another variant of IO prompting named Few-shot, this technique combines the task instructions with examples of the problem and the desired solution [4]. Another prompting technique is Chain-of-Thought (CoT); it is used

in challenging tasks for LLMs, such as complex arithmetic problems, by adding examples that include intermediate reasoning steps [15].

Research conducted in [5,8,11] evaluated the potential to create domain models from textual descriptions using IO prompting. Camara et al. used zero-shot prompting to create UML class diagrams with few syntactic errors, however, the worst results were found when the model required abstractions, such as using inheritance instead of attributes or creating association classes [8]. Fill et al. used GPT-4 to create domain models for Entity Relationship diagrams for conceptual modeling, BPMN diagrams for business processes, and Heraklit models for embedded systems by providing one example of the desired output [11]. Another research experimented with the Few-shot technique using between 2 and 4 examples for the recommendation of concepts for different domain contexts, and to assist with static and dynamic domain modeling [5]. Additionally, Chen et al. compared the performance of suggested model elementes compared to a reference solution using the Few-shot technique with one or two examples to generate domain models, and concluded that adding two examples does not improve the performance in comparison with one example [6]. These experiments also used the CoT prompting with one example that illustrates the intermediate reasoning steps where the domain description was divided into sentences, and for each sentence the model elements are recognized. These experiments compared IO prompting with CoT prompting, and surprisingly, the use of the intermediate reasoning steps in CoT showed no improvement in performance to recommend model elements [6]. Using prompt techniques has shown that the recommendation of domain model elements has some limitations because LLMs struggle to classify relationships [6], to recommend complex constructs such as association classes [8], and using more examples in the prompts does not increase the performance of the elements suggested [6].

## 3 Background: Tree-of-Thoughts Framework

The Tree of Thoughts (ToT) is "a paradigm that allows LMs to explore multiple reasoning paths over thoughts" [17]. The thoughts are used as intermediate steps towards a problem solution. The tree representation is given because in each intermediate step there are alternative thoughts as shown in Fig. 1 that are evaluated to select the most promising to continue with the problem solution. To implement the framework, first the LLM acts as a Thought generator to create a set of thoughts that correspond to a partial solution of the problem. Then, the LLM is used as State evaluator to self-evaluate the thoughts and choose the most promising option to solve the problem [17]. The following four questions are to be analyzed in the problem to apply the ToT framework:
(1) *How to decompose the intermediate process into thought steps*: The thought decomposition requires us to design the intermediate thought steps. These should be promising and diverse, in order to evaluate the best prospects to achieve a solution. Some recommendations for the thoughts are that it is not too 'small' (e.g. a word or sentence) that cause it lack of diversity, and it is not too 'big' (e.g. a book) that makes it difficult to evaluate the coherence [17].

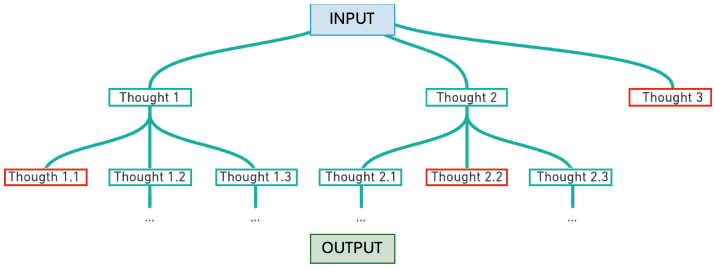



**Fig. 1.** Representation of ToT approach, the green boxes are the intermediate thoughts used for the problem solution. The number of thoughts is $k$=3 and the number of promising thoughts to explore is $b$=2. The red boxes indicate the thoughts not explored. (Color figure online)

(2) *How to generate potential thoughts from each state*: There are two possible strategies to generate the intermediate thoughts, and the best option will depend on the problem to be resolved.

- Sample: The same prompt creates diverse outputs, and each one can be used independent thoughts. This strategy is preferred when the thought is a paragraph, and the samples are diverse [17].
- Propose: The prompt proposes multiple thoughts in the same output, and we require to avoid duplicated thoughts in the same context. This is used when the thought is constrained to word or phrase. For example, in the Crosswords game, we require with one prompt multiple words, and each word is a thought towards a solution [17].

(3) *How to heuristically evaluate states*: The evaluation of states determines which thoughts are the best prospects to continue with the solution of the problem. For evaluation, two strategies are suggested:

- Value: The LLM prompt evaluates the thought with a scalar value (e.g., 1–5) or classification (e.g., complete/partial/incomplete) [17]. It is suggested when the valuation helps to decide the best choice.
- Vote: The prompt compares the different thought options and votes for the best promising one. This strategy is preferred to use when comparing partial solutions [17].

(4) *What search algorithm to use*: The search algorithm is used to explore promising thoughts identified by the state evaluator. The algorithms used in the experiments in [17] are as follows:

- Breadth-first search (BFS): In every step of the algorithm, the $b$ most promising thoughts in each tree level are maintained to continue exploring in the next level. The number of $k$ generated thoughts is reduced to $b$ most promising thoughts, as shown in Fig. 1. Both $k$ and $b$ are configurable parameters in the algorithm defined in [17].

– Depth-first search (DFS): In this algorithm, the most promising thought among the $k$ thoughts proposed is explored until either a final result of the problem is achieved, or we reach an intermediate state that is deemed impossible to achieve a solution by the state evaluator[1]. In the latter case, the state is pruned and the algorithm backtracks to the parent state to explore the next most promising thought.

## 4 A ToT Framework Setup for Domain Modeling

In this section, we review our approach to implement the Tree of Thoughts for task decomposition in domain modeling, selection of thought generator and evaluator strategies, and search algorithm. We illustrate this section with an example to create UML class diagrams.

### 4.1 Domain Modeling Task Decomposition

Figure 2 represents the tree structure identified to create a UML class diagram with five modeling tasks: (1) identifying classes and attributes, (2) identifying associations among classes, (3) refining associations to containments, (4) identifying inheritance, and (5) refining associations to association classes.

Proposals to create UML class diagrams from natural language texts use two steps as implemented in [3] and [14]. The selection of model elements is applied sentence by sentence in the domain descriptions and starts with class identification, then continues with relationship classification. Our proposal will use these two intuitive steps to separate classes from relationships; however, we will preserve the entire domain description as input for ToT because previous experiments using CoT with sentence decomposition [6] did not show improvements in performance for suggestion of model elements.

Furthermore, in the rule-based NLP approach [3], relationships are classified using different extraction rules per relationship type. Moreover, the performance of LLMs on the relationship identification task is the worst compared to class identification [6]. For those reasons, our approach starts with the identification of classes and attributes, then uses this intermediate thought to identify and classify the relationships for those classes. We propose to divide it into three tasks: association, containment, and inheritance. We identify that recognition of relationships performed by human modelers follows this intuitive approach starting by recognizing associations and then classifying containment or inheritance.

The thought decomposition will be completed by the identification of the association classes. As reviewed in the previous section, the association class is an abstraction in the UML class diagram where GPT does not produce good results [8]. The UML specification states that this abstraction is a class and an association. For that reason, we choose to separate it in a step after these two thoughts are completed.

[1] The state evaluator evaluates the progress made towards solving the problem, serving as a heuristic for the search algorithm to determine which states to keep exploring and in which order [17].

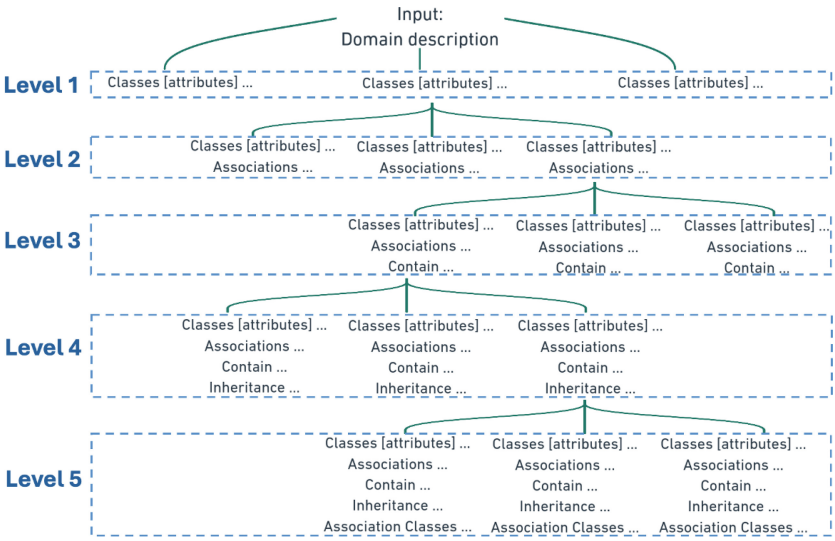



**Fig. 2.** Representation of the thought decomposition in the tree of thoughts. The root node represents the domain description as the input to generate the thoughts. The thoughts are alternatives of model elements. The blue boxes represent all the thoughts in the same level, and the best thought selected is used to continue with the model creation. (Color figure online)

### 4.2 Generator Strategy

After defining the task decomposition, we evaluated the generator strategy. The purpose of the generator prompt is to create diverse and promising thoughts. As discussed earlier, two strategies are possible: the sample strategy is used when the problem space is rich (e.g. the LLM provides diverse responses for the same prompt); and the propose strategy is better when the problem space is more constrained (e.g. the thought is a word or a line) [17]. We choose the sample strategy for the generator prompt because we have experimented that diverse model elements are suggested even with the same prompt. The reason for this behavior is that LLMs are not deterministic and different domain models are generated for the same prompt [8]. Our proposal is to generate $k$=3 thoughts per tree level as shown in Fig. 2, so that the evaluator has a sufficient variety of options to select from.

### 4.3 Evaluator Strategy

In the case of the evaluator prompt, our purpose is to assess the proposed modeling elements and choose the best options to continue with the domain model creation. In this scenario, we discarded the value strategy because it requires to score each individual thought with a scalar value, which is difficult to obtain objectively even for human modelers [6]. Therefore, the vote strategy [17] is

selected, where we ask the LLM to act as a domain modeler expert and select the best proposal among the generated options. The LLM votes 5 times, and the option with the highest number of votes is maintained at each level. Having a higher number of votes (i.e., 5) compared to the number of thoughts (i.e., 3) reduces the chance of ties.

### 4.4 Search Algorithm

To complete the definition of the ToT framework, we evaluated the BFS and DFS algorithms. The BFS algorithm self-evaluates the thoughts and maintains the $b$ most promising ones at each level. Then, it continues the exploration with the best thoughts identified. In the case of DFS, the most promising thought is explored until a final solution is achieved. When such a solution is not possible to achieve, the algorithm backtracks to the next most promising thought [17]. In our proposal, we select the BFS algorithm with $b = 1$. Namely, for each level, we always select the best proposal and continue with it for the creation of the domain model afterwards, as shown in Fig. 2. BFS allows us to build upon the best previous thoughts without the need to backtrack.

## 5 A DSL to Configure ToT Framework Setups

The decomposition of the modeling tasks defined in Sect. 4 is an example that modelers could modify based on the purpose, size, and complexity of their modeling process. Using a DSL for complex techniques is advised [11] because it simplifies running experiments without the need to modify the ToT underlying code. The role of the DSL is to facilitate the configuration and automate the execution of the ToT, based on the task decomposition identified by the modeler.

### 5.1 DSL Abstract Syntax

Figure 3 presents the metamodel of the DSL, comprising two packages: the problem space package that contains concepts pertinent to the domain modeling problem, and the solution space package mainly to setup the ToT configuration as a solution to the domain modeling problem.

The problem space is characterized by the modeling problem with a description of the domain being modeled and a purpose of modeling, e.g., to create a UML class diagram. The modeling problem is then decomposed into a sequence of tasks, each of which is identified by a name and a task description. Moreover, each task contains one or more assessment criteria against which model elements proposed by the task should be evaluated. Finally, one also needs to specify the modeling notation to use for representing the final domain model.

The ToT configuration is set up using concepts from the solution space. Basically, one instantiates a tree by specifying the number of levels, the number of samples (thoughts) to create at each level, and the number of vote to cast at each level to select a winning thought. The problem space specification informs such

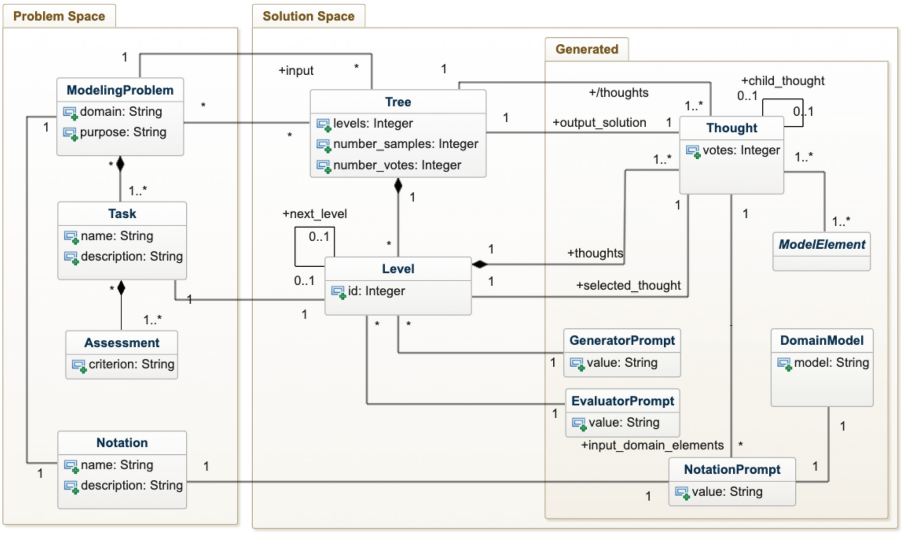



**Fig. 3.** DSL to facilitate the configuration and automate the execution of the ToT framework for domain modeling tasks.

a tree configuration. For example, the levels of the tree should be in accordance with the tasks, whereby the task description constitutes an integral input for designing the generator prompt of the level, and the assessment criteria are to be incorporated in the evaluator prompt. Moreover, both the domain description and the modeling purpose are repeated throughout all the generator prompts.

A part of the solution space concepts, grouped in the generated sub-package, are generated during the ToT execution (instead of instantiated by the user). These include the thoughts, i.e. intermediate (partial) domain models, created by the LLM, and the selected winning thought of each level as evaluated by the LLM. Moreover, all prompts, including those for generating the thoughts, for evaluating the thoughts, and for transforming the final thought into the modeling notation of choice, are also generated following predefined templates. More specifically, the generator prompt of a level incorporates the description of the task tackled by the level and the winning thought of the previous level, the evaluator prompt enumerates the assessment criteria specified for the corresponding task, and the notation prompt refers to the named notation and its description.

For example, Fig. 4 shows the prompt templates (in the first row) and the corresponding generated prompts for the task to identify association classes, the fifth level of the tree presented in Fig. 2. The prompts are available on GitHub[2].

[2] https://github.com/BESSER-PEARL/dsl-tot-dm/blob/main/dsl/prompts.py.

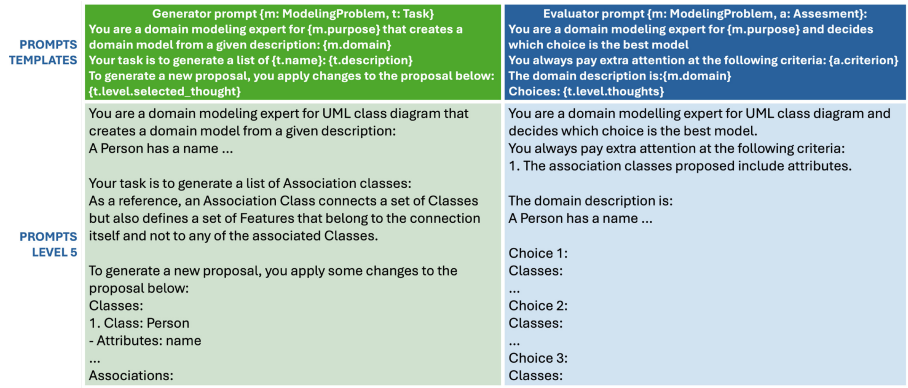



**Fig. 4.** Templates of generator and evaluator prompts, and example prompts for fifth level of the ToT.

### 5.2 DSL Concrete Syntax

We provide the concrete syntax designed with the grammar of TextX [9] for the metamodel defined in Sect. 5.1. The grammar shown in Listing 1.1 allows customization of the ToT for modeling tasks.

```
Model: tree=Tree  problem=Problem
tasks=Task+ notation=Notation;
Tree: ''tree:
  ''levels:'' levels=INT
  ''number_samples:'' number_samples=INT
  ''number_votes:'' number_votes=INT;
Problem: ''problem:''
  ''domain:'' domain=STRING  ''purpose:'' purpose=STRING;
Task: ''task:''
  ''level:'' level=INT  ''name:'' name=STRING
  ''description:'' description=STRING
''assessments:'' assessments=STRING+;
Notation: ''notation:''
  ''name:'' name=STRING  ''description:'' description=STRING;
```

**Listing 1.1.** Grammar for the concrete syntax definition

We illustrate the concrete syntax using the example ToT configuration presented in Fig. 2. As can be seen in Listing 1.2, the tree consists of 5 levels. For each level, the generator prompt is executed 3 times to create 3 sample thoughts, and the evaluator prompt is executed 5 times to vote for the most promising thought. The next step is to describe the modeling problem by specifying the purpose of the modeling process and the textual description of the domain. The latter can be provided by either a string or through a path to a text file that contains the domain description. In our example, the description is given by a text file.

After that, intermediate modeling tasks are specified, with the level at which a task is, the task name, and the task description that defines the modeling knowledge relevant to the task, for example, what an association class is. Furthermore, each task requires at least one criterion to evaluate its thoughts. For example, the fifth level concerns the creation of association classes. The classes proposed by the LLM will be evaluated against one criteria: "The association classes proposed include attributes.".

```
tree:
    levels: 5
    number_samples: 3
    number_votes: 5
problem:
    domain: ``./domain/exercise1.txt''
    purpose: ``UML class diagram''
task:
    level: 1
    name: ``Classes''
    ...
task:
    level: 2
    name: ``Association''
    ...
task:
    level: 3
    name: ``Containment''
    ...
task:
    level: 4
    name: ``Inheritance''
    ...
task:
    level: 5
    name: ``Association classes''
    description: ``As a reference, an Association Class connects
    a set of Classes but also defines a set of Features that belong
    to the connection itself and not to any of the associated Classes.''
    assessments:
        ``The association classes proposed include attributes.''
notation:
    name: ``Plantuml code''
    description: ``Create the PlantUML ...
association classes use the format: (<Source_Class> ,
    <Target_Class>) .. <Association_Class>''
```

**Listing 1.2.** Concrete syntax for Tree, Problem, Tasks, and notation

Listing 1.2 describes that the first task is to identify classes. Then, the tasks from level 2 to 4 are to identify the relationships divided into three tasks: Association, containment, and inheritance. In level 5, the task is specified for association classes. Our purpose using this approach is to influence the association class identification with two key thoughts: 1) the prompt has information about classes and associations identified in the upper level and 2) by adding modeling knowledge with the description of association classes.

The last step is to specify the modeling notation (e.g., PlantUML), in which the thoughts created in the tree should be represented. We describe the syntactic representation of association classes in PlantUML in the description.

## 6 Validation

In this section, our aim is to experiment with the ToT prompting technique to create UML class diagrams that include association classes, which LLMs struggle to recommend. The use of the DSL, based on our own experience, has helped reduce the complexity of setting up the ToT configuration and automating the testing process to a great extent.

### 6.1 Experiment Setup

Our experiment dataset is composed of five domain descriptions and five corresponding reference solution domain models expressed as UML class diagrams. They are collected from various sources: the Theaters and Robots domains used by Camara et al. [8], the Person and Resources domains from a university modeling course, and the e-Commerce and Courses proposed by us. The reference models contain 35 classes, 40 attributes, 24 relationships (including both association, containment, and inheritance), and 9 association classes in total.

To run the experiments, we use GPT-4 as the example LLM, which is the model with the best results for the creation of domain models according to [6]. We use the five-level ToT configuration presented in Sect. 4 in our experiments, namely to identify classes and attributes, associations, containment relationships, inheritance relationships, and association classes.

### 6.2 Evaluation Criteria

For the evaluation of the models, we compare the output in PlantUML notation and consider a correct prediction if the element of the suggested model is semantic equivalent to the reference model, including elements with different names. An example of similarity is shown in Fig. 5b, the association class and attributes names are different from the reference model; however, it is equivalent because the source class, target class, association class and attributes have the same meaning in the domain model. In contrast, in Fig. 5c, the association class is incorrect because the class session is used as the target class and has a different meaning in the domain model compared to the reference model.

The criteria for the evaluation of classes and attributes are similar; we consider a correct prediction the use of equivalent names. For attributes, we are not evaluating the data types. In the case of relationships, the correct prediction should match the lower and upper cardinality; if the lower cardinality is not specified in the reference model, we consider only the upper cardinality.

```
@startuml
...
class Session{
  int timeBegin
  int timeEnd
}
...
(Play, Participant) .. Session
...

@enduml
```

(a) Reference solution

```
@startuml
...
class Play_Session_Participant {
    +Session start time
    +Session end time
}
...
(Play , Participant) ..
Play_Session_Participant
...
@enduml
```

(b) Equivalent

```
@startuml
...
class Act {
    - role: string
}
...

(Participant, Session) .. Act
...

@enduml
```

(c) Not equivalent

**Fig. 5.** Comparison of association classes in PlantUML syntax

To evaluate the experiments, we used precision, recall, and the F1 score. For precision, we consider equivalent predictions as true positives ($TP$), and not equivalent predictions as false positives ($FP$). The formula for precision is:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

For recall, we use the $TP$ and the elements not predicted as false negatives ($FN$). The calculation is performed using the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Finally, we use the F1-score formula below:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

### 6.3 Experiment Results

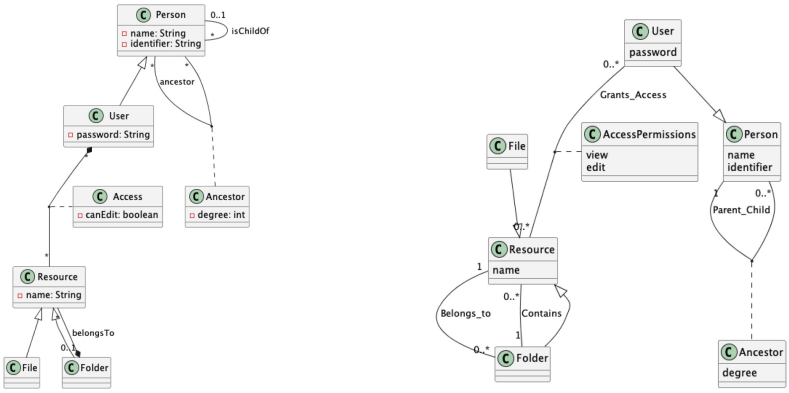
Our aim with these experiments is to compare the performance of the ToT framework to recommend model elements with the reference solutions. Table 1 shows that the performance is better to suggest classes and association classes between all the model elements. For attributes and relationships, the performance of the recommendations decreases. Although we have specific intermediate steps for association, containment, and inheritance, the result for relationships is the worst among all the model elements. The issue with relationship recommendation is a well-known issue that was reviewed in the previous sections. Finally, the attribute elements are not an intermediate step and are suggested with the classes, which appears to cause lower performance using ToT.

The results reported in Table 1 show that ToT has the best performance for recommendation of classes and association classes with 0.895 and 0.824 for the F1 score, respectively. The precision and recall performance in these elements are very similar, and no major gap is identified. However, when comparing the performance of the attribute element recommendation, there is a major difference

**Table 1.** Result of the performance of ToT for UML class model elements

| | Precision | Recall | F1-score |
|---|---|---|---|
| Class | 0.829 | 0.971 | 0.895 |
| Attribute | 0.545 | 0.900 | 0.679 |
| Relationship | 0,227 | 0,417 | 0,294 |
| Association Class | 0.875 | 0.778 | 0.824 |

between precision and recall, specifically with a low precision score of 0.545 compared to the recall of 0.9. This result indicates that various suggested attributes are not included in the reference model. In Fig. 6a, we have an example of a model that contains two association classes: access and ancestor. These classes are correctly recommended at the fifth level and are included in the model shown in Fig. 6b. We found that both elements were suggested for the 3 thoughts at that level, but with slight name differences, such as genealogy.




(a) Reference solution (b) ToT with 5 levels

**Fig. 6.** Person and Resource domain model used in the experiments

Regarding relationship recommendations, the performance of the F1 score is lower compared to the other model elements with 0.294. The performance for both precision and recall are the lowest compared with the other model elements. Similarly to attributes, the lower precision indicates that the relationships recommended are not part of the reference model, and the lower recall indicates that various relationships are not recommended. The reference solution in Fig. 6a has a containment relationship between the folder and resource classes; however, the ToT suggests two associations, as shown in Fig. 6b. We found that in the third level of the ToT, duplicated relationships are suggested between Resource and Folder for all the thoughts, which cause the error in the final model.

### 6.4 Threats to Validity

*Equivalent Model Elements for Association Classes.* It is possible to represent association classes in a syntactically distinct but semantically equivalent way replacing them with one class and two associations for the source and target classes. We are not considering these equivalent representations in the evaluation of our experiments.

*LLMs are not Deterministic.* LLMs are generative models that produce different results with similar prompts. Because it is not deterministic, we can generate different thoughts using the same generator prompt; however, this also affects the evaluator votes that can choose two different thoughts as the best option with the same prompt.

*Reference Models in the Dataset.* Due to the lack of data with domain description and reference models, we have used exercises from different sources: previous papers, a university modeling course, and created by us.

## 7 Discussion

*Performance.* Our experiments show that using ToT enables GPT-4 to create complex modeling constructs (e.g., association classes) as part of the final domain models. ToT performs the best at identifying classes and association classes, compared to identifying other model elements. Our conjecture is that the intermediate results of preceding steps, namely identification of the best fit classes and associations, provide insights that extends beyond the domain description, which ToT was able to leverage towards recommending suitable association classes. This result is encouraging to explore more configurations in future research for task decomposition for other modeling processes, such as behavioral diagrams.

To address the performance of ToT to recommend attributes, we envision that extending the task decomposition with an additional level dedicated to attribute generation and evaluation would help. Regarding the relationships, we confirm that GPT-4 recommendations are not reliable and ToT technique alone can not resolve this issue. As a consequence, a combination of ToT with other prompt techniques, such as using few-shots at the levels regarding relationship generation would be the next target of investigation, especially for identifying associations and containment relationships where LLMs do not differentiate properly.

*Generator Prompts.* We have reviewed the diversity of the thoughts created by the generator prompt, and observed that fewer diversity exists for thoughts related to inheritance in comparison with association and containment. For those model elements that have less diverse thoughts, we recommend using the Few-shot or CoT prompting techniques for the recommendation of model elements.

*Evaluator Prompts.* The number of votes selected was 5 to avoid ties while evaluating thoughts; however, we have observed that ties occur for some thoughts in

association classes that have two different proposals. This result indicates that in certain domains the result of the evaluator prompt does not recognize the difference between two different thoughts, and the assistance of a human expert will be useful to choose the best option. For future research, we suggest including human modeling experts in the evaluator role to select the best thoughts.

*Costs.* Use of the ToT framework involves a higher cost because more prompts are used compared to other techniques. We suggest to experiment with this framework for complex modeling tasks where other prompting techniques do not achieve good results. Moreover, the DSL is an open source tool that can be extended to experiment with open-source LLMs to reduce such cost.

## 8 Tool Support

We implemented an open-source DSL tool on GitHub[3]. It allows the definition of ToT configurations for domain modeling and the execution of them to generate and evaluate thoughts, as detailed in Sect. 5. The dataset and results of the experiments are included in the same repository. Additionally, the repository includes additional examples that use ToT configurations with other modeling languages, such as Entity-Relationship and UML activity diagrams.

The DSL is defined in Python using TextX [9]. The Python template shown below is used for the code generation that instantiates the classes from our DSL, these classes use the ToT framework library [17] to execute the generator and evaluator strategy with the BFS algorithm. To use the DSL, it is required a subscription to OpenAI API or Azure OpenAI API; however, it is possible to extend the DSL to use it with other LLMs.

```
...
def run():
    problem = ModelingProblem(levels = {{tree.levels}}, purpose = ‘‘{{problem.
    purpose}}’’, description = ‘‘‘‘‘‘{{ problem.domain }}’’’’’’)
    tree = Tree(number_levels = {{tree.levels}}, generator_samples = {{tree.
    number_samples}}, evaluator_votes %= {{tree.number_votes}})
    ...
    output = tree.execute(logName)
    ...
```

**Listing 1.3.** Python template used for code generation

## 9 Conclusion and Further Work

Creating domain models is a complex task that demands domain experts and modelers a profound understanding of the domain to conceptualize the elements that will be included in the model. The creation of domain models with prompting strategies has limitations, and we support the idea of decomposing the process into tasks and use the ToT framework to improve the recommendation of

[3] https://github.com/BESSER-PEARL/dsl-tot-dm.

model elements. Furthermore, we propose the use of a DSL that orchestrate and executes the prompts needed to generate and evaluate model elements in various thoughts. Using GPT-4, our ToT approach obtained better results for class and association classes; however, for attributes and relationships, some recommendations are not accurate for the domain (low precision).

The potential of our ToT approach resides in its design to generate model recommendations by recommending alternatives of model elements in intermediate thoughts and self-evaluate them with specific criteria. As future work, we plan to combine the prompting techniques for domain modeling, e.g. using Few-shot examples for the generator prompts. Moreover, to facilitate the use of the DSL, we will add configuration templates to represent different modeling processes that modelers can use as basis to define their own configurations.

## References

1. UML Specifications. (2015). https://www.omg.org/spec/UML/2.5/PDF/. Accessed 3 May 2024
2. Almonte, L., Guerra, E., Cantador, I., De Lara, J.: Recommender systems in model-driven engineering: a systematic mapping review. Softw. Syst. Model. **21**(1), 249–280 (2022) https://doi.org/10.1007/s10270-021-00905-x
3. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Extracting domain models from natural-language requirements: approach and industrial evaluation. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 250–260. ACM, Saint-Malo (2016). https://doi.org/10.1145/2976767.2976769
4. Brown, T., et al.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
5. Chaaben, M.B., Burgueño, L., Sahraoui, H.: 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), pp. 7–12. IEEE, Melbourne (2023). https://doi.org/10.1109/ICSE-NIER58687.2023.00008
6. Chen, K., Yang, Y., Chen, B., Hernández López, J.A., Mussbacher, G., Varró, D.: 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), pp. 162–172. IEEE, Västerås (2023). https://doi.org/10.1109/MODELS58315.2023.00037
7. Chen, P.P.S.: The entity-relationship model-toward a unified view of data. ACM Trans. Datab. Syst. **1**(1), 9–36 (1976). https://doi.org/10.1145/320434.320440
8. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. Softw. Syst. Model. **22**(3), 781–793 (2023). https://doi.org/10.1007/s10270-023-01105-5
9. Dejanović, I., Vaderna, R., Milosavljević, G.: Vuković,: Textx: a python tool for domain-specific languages implementation. Knowl.-Based Syst. **115**, 1–4 (2017). https://doi.org/10.1016/j.knosys.2016.10.023

10. Feltus, C., Ma, Q., Proper, H.A., Kelsen, P.: Towards AI assisted domain modeling. In: Reinhartz-Berger, I., Sadiq, S. (eds.) Advances in Conceptual Modeling. LNCS, vol. 13012, pp. 75–89. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88358-4_7
11. Fill, H.G., Fettke, P., Köpke, J.: Conceptual modeling and large language models: impressions from first experiments with ChatGPT. In: Enterprise Modelling and Information Systems Architectures (EMISAJ), pp. 3:1-:15 (2023). https://doi.org/10.18417/EMISA.18.3, https://emisa-journal.org/emisa/article/view/318. Artwork Size: 3:1–15 Pages Publisher: Enterprise Modelling and Information Systems Architectures (EMISAJ)
12. Frederiks, P.J.M., van der Weide, T.P.: Information modeling: the process and the required competencies of its participants. Data Knowl. Eng. **58**(1), 4–20 (2006). https://doi.org/10.1016/j.datak.2005.05.007
13. Mussbacher, G., et al.: Opportunities in intelligent modeling assistance. Softw. Syst. Model. **19**(5), 1045–1053 (2020). https://doi.org/10.1007/s10270-020-00814-5
14. Saini, R., Mussbacher, G., Guo, J.L.C., Kienzle, J.: DoMoBOT: a bot for automated and interactive domain modelling. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–10. ACM, Virtual Event Canada (2020). https://doi.org/10.1145/3417990.3421385
15. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems, vol. 35, pp. 24824–24837. Curran Associates, Inc. (2022). https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
16. Wilmont, I., Hengeveld, S., Barendsen, E., Hoppenbrouwers, S.: Cognitive mechanisms of conceptual modelling. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) Conceptual Modeling, pp. 74–87. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41924-9_7
17. Yao, S., et al.: Tree of thoughts: deliberate problem solving with large language models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 11809–11822. Curran Associates, Inc. (2023). https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf