Task Decomposition and Partial Solution Model: Automated Domain Modeling Assistant with Large Language Models

Noé Jager

[a] *University of Montréal, Montréal, Canada*

Abstract

Domain modeling is a creation process to create an abstract representation of a solution domain, answering to a problem domain. This usually involves an expert domain modeler spending a lot of time incrementing the model as he gets feedback from a domain expert. This is a time costly process and requires a domain expert. With the rise of Large Language Model assistants, we can try to implement systems building domain models in less time with less to no human intervention. We built a Python Automated Model Assistant connecting to OpenAI and Gemini models to built metamodels using Few Shots, and for the first time to my knowledge, Task Decomposition methods and the option to add Partial Solution Models to the prompt. The results shows that it is possible to attain F1 Scores of 0.90 for classes, 0.82 for attributes, and 0.79 for relations by adding a 40% Partial Solution Model to the prompt.

Keywords: Model Driven Engineering, Domain Model, Prompt Engineering, Large Language model, automated assistant

1. Introduction

Context: In Model driven Engineering (MDE), Domain modeling is the process of creating an abstract version of the solution entities. It requires a lot of time and effort through an iterative process, and requires the presence of a domain expert and of a domain model designer expert.
Since the release of ChatGPT in November 2022 (3), Large Language Models (LLM) have gain a lot of fame in completing most Natural Language (NL) tasks anyone could think of. They uses a new type of Deep Learning

model using self attention transformers and an extensive dataset from the web. Since then, researchers have worked on finding a way to use LLMs to build domain models. By transforming the design of a domain model as a NL tasks, those models have a wide range of domain knowledge and thus could bypass the need for a domain expert, as well as reducing the time of design.

Problem statement: This paper aims at creating an automated domain modeling assistant using LLM model APIs and upgrading previous researches model generated performances.

Contribution: This paper contributes by creating a fully automated and user friendly Python domain model assistant (12). The specific contributions of this paper are:

1. A fully automated Domain Model design assistant in Python, available for anyone to use with specific settings, with export options such as NL, ecore model and custom json model,
2. The evaluation of a new prompt engineering method called Task Decomposition with two levels either manual decomposition by sentence or automated with LLM task generation from a textual description. Task decomposition and generation have made their proof in other LLM prompt engineering researches but have never been used for domain model generation to my knowledge,
3. The addition and evaluation of a assistant feature where a user can add a pre-existing model to the LLM.

Paper Organization:

- **Section 2: Background** presents the methodology of the paper by presenting concepts like Domain Modeling, and evaluation as well as the prompt engineering settings used and why.
- **Section 3: Approach** presents the implementation of the Automated Model assistant, the inputs and outputs, and some flowcharts of scenarios the implementation follows depending on the settings used.
- **Section 4: Evaluation** presents how the evaluation data was measured, what are the research questions, and show in details the estimated means, contrasts and p-values of settings and interactions with figures and tables.

- **Section 5: Discussion** discuss the results and makes the relation with related works.
- **Section 6: Conclusion** concludes and introduce future works.

2. Background

2.1. Domain Modeling

Model Driven Engineering (MDE) is a methodology that has proven useful. Building a Domain Model is similar to creating blueprints for your solution system. Thus, you elevate your solution to a higher degree of abstraction, making it easier to reason and facilitating the sharing of domain knowledge with domain specialists which do not need to know programming languages to provide input.
However, there are some constraints to domain modeling. It is heavily dependent on a user expertise level in modeling and this domain, as well as being an iterative, and time consuming process. [(1), (2)]

2.2. Large Language Models

Since its release in 2022, ChatGPT has revolutionized the way people work and learn. Using a new technology of self-attention mechanisms and transformer-based models, Large Language Models have broken all records in NL tasks with their omniscient knowledge from internet. [(3)]
In the field of Model Driven Engineering, recent researches have shown that LLM, like GPT 3.5 and GPT4 from OpenAI, have a promising future in the design of Domain Models. They bypass the two constraints explained above with their quick execution and their knowledge of various domains.
However, by using simple prompt engineering techniques like Few Shots, they suffer from a low recall reducing the overall performances. Moreover, the Chain-Of-Thought techniques doesn't make any significant changes on the output. [(4)]

2.3. Passing a Domain Description to a LLM

Usually domain models are created from a textual description of the problem domain and are then transformed into models usually using languages with a graphical concrete syntax like UML or Ecore.
LLMs take as input text, it means it is possible to provide them the textual description a domain modeler would usually use. However, this should be treated as a text generation problem since they also output texts.

2.4. Prompt Engineering methods

Prompt Engineering is a new concept that appeared with Large Language Models. Since the output from a LLM depends heavily on the prompt passed, many methods have seen the day to try and get the best performances from them [(5)]. This research uses the following three methods:

- Few Shots: Method for when you have a limited number of examples to pass to an LLM to teach him a pattern. Before passing the final prompt to the LLM, you first have an exchange passing an prompt example and the expected output.
- Task Decomposition: Method consisting in dividing complex tasks in smaller simpler ones. This is particularly efficient in tasks requiring multiple reasoning steps. [(6)]
- Partial Solution Model (PSM): Depending on the specific task for the LLM, a partial solution can be provided as input for the model to complete. This approach narrows the range of possible interpretations for the task by using the LLM as a domain model completion assistant, effectively guiding the model's focus to complete or extend the provided solution. [(9)]

2.5. Domain Modeling Evaluation

This paper follows the evaluation method from [(4)]. Using a solution model, their method is to compute the F1-score of a domain model by:

1. Categorizing all model elements (Class, Attribute, Relation) from both the output and solution models with 1.
2. Assign points to all elements as 1.
3. Compute the precision, recall and F1-score for each element type using 2, 3 and 4.

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

$$Precision_c = \frac{\sum_{i=0}^{i=m} S(C_i)}{m} \tag{2}$$

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

Table 1: Evaluation Categories table

$$Recall_c = \frac{\sum_{i=0}^{i=n} S(C_i)}{n} \tag{3}$$

$$F1_c = \frac{2 \times Precision_c \times Recall_c}{Precision_c + Recall_c} \tag{4}$$

Using this method, we can grade and evaluate the generated models for each of its element type. Another considered way would have been to compute the model distance [(11)].

3. Approach

This section describes the implementation of a Domain Model Assistant LLM. All of this is done in a python application well documented, user friendly available on GitHub (12). All the Problem Descriptions and their solutions are collected from this previous research [(4)].

3.1. Approach settings

This section details the 4 user's parameters in the system.

- LLM Model: Which model to use between:
    1. gemini-1.5-flash: A free and fast option suitable for lightweight tasks and initial exploration.
    2. gpt-3-turbo: A cost-effective choice offering a balance between speed and quality.

3. gpt-4: The most advanced model, providing superior accuracy and depth at a higher cost and slower speed.

- Few Shots: Which and how many Domain Model input and solution the user wants the LLM to learn from.
  To do so the user defines it as follow: "2shot_dsl1_dsl2".

- Task Decomposition Method: If the user wants to decompose his textual domain description in smaller sub tasks, and how.
  There are 3 options available:
  - "" means the user wants to pass the full domain description at once,
  - "manual" means the user wants to divide the domain description for each line.
  - "auto" means the LLM gets a first call where he has to define a list of 10 CRUD operations another model will have to follow to generate the domain model.

- Partial Solution Model: It is possible to pass an initial model to the LLM for him to work on.

3.2. Encoding of the inputs/outputs of the LLM

The application stores and handles model in json format 1. This format allows for a simple ecore model of enumerations and classes. Each class have:

- a name,
- a boolean abstract state,
- attributes of types: string, integer, boolean, float, double, time, date, array, and other complex types like other classes or enumeration; Also attributes can have a default value.
- three types of relations: *Associations*, *Containments* and *Inheritance*. *Associations* and *containments* relations have multiplicities.

Figure 1: The Json format of the solution models and outputs of the LLM

```
{
        "enum": [
            {
                "name": "EnumName",
                "values": ["Value1", "Value2", ...]
            },
            ...
        ],
        "class_": [
            {
                "name": "ClassName",
                "abstract": true | false,
                "attributes": [
                    {
                        "name": "attributeName",
                        "type": "string" | "int" | "boolean" | "float" | "double" | "time" | "date" | "array" | "customType"
                    },
                    ...
                ],
                "contains": [
                    {
                        "to": "ContainedClassName",
                        "mul": "0..1" | "1" | "*"
                    },
                    ...
                ],
                "associate": [
                    {
                        "to": "AssociatedClassName",
                        "mul": "0..1" | "1" | "*"
                    },
                    ...
                ],
                "inherit": ["SuperClassName", ...]
            },
            ...
        ]
}
```

3.3. Generation of a domain model

The generation process of the domain model varies on the three settings passed by the user:

- If the user adds shots for LLM, the system will iteratively inputs the textual description and guide the model answer with the correct solution model. At the end, it prompts the textual description/tasks/PSM to the model and exports the model to ecore/json. See diagram 2

- If the system is in auto mode, it begins by taking the textual problem description as input. The LLM generates a list of 10 subtasks. If the system is in manual mode, it separates the textual description in subtasks. These subtasks are saved and inputted iteratively to the LLM model along with the solution model. For each subtask, if there are more subtasks to process, the loop continues. Once all subtasks are

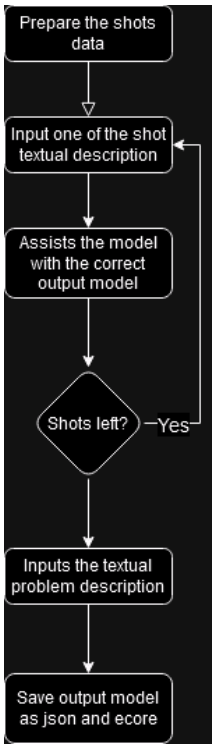

Figure 2: Scenario how the assistant uses shots for the prompt.

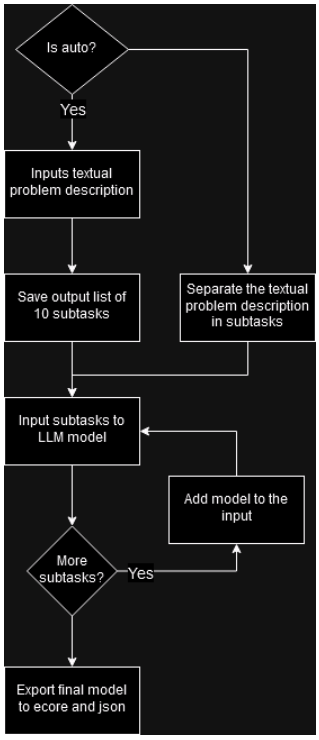

Figure 3: Scenario how the assistant uses Task Decomposition for the prompt.

processed, the final model is exported in both ecore and JSON formats. See 3

- If the user adds a Partial Solution Model, the system starts by loading the Partial Solution Model (PSM). The PSM is then added to the input, after which the output model is generated. Finally, the output model is exported in both ecore and JSON formats. See 4

Those three scenarios are not exclusive to each other and can work simultaneously. For example, a call can have be a 2 shots, Auto Task Decomposition and have a PMS and thus do all the following above.
The full System state diagram is available on the GitHub repository (12).

4. Evaluation

This section details the process of grading of the generated model to do a detailed evaluation of the parameters through Research Questions detailed

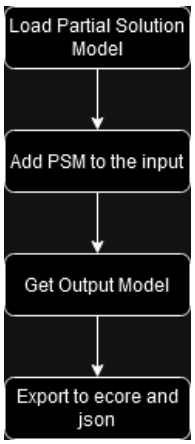

Figure 4: Scenario how the assistant uses a PSM for the prompt.

in Section 4.2.

4.1. Evaluation settings

Table 2: Collected domain models, their partial solution models (PSM) and their numbers of modeling elements

	HBMS			H2S			TSS		
PSM	Class	Attribute	Relation	Class	Attribute	Relation	Class	Attribute	Relation
Full Solution	15	21	18	12	15	20	11	11	18
3	6	10	10	5	9	10	5	6	7
2	5	7	7	4	8	8	4	4	7
1	4	5	3	3	4	7	3	3	6

Table 3: Completeness Ratios across the Partial Solution Models (PSM) : intended, obtained one, modeling elements ratios.

PSM	Intended	Real	Class	Attribute	Relation
3	0,4	0,4823	0,4211	0,5319	0,4821
2	0,33	0,3830	0,3421	0,4043	0,3929
1	0,25	0,2695	0,2632	0,2553	0,2857

The grading of the models is done on an excel sheet using EMF Compare as a support tool to help see similarities and differences between the two models. The grading method in detail has already been described earlier in

section 2.5.
Two datasets are collected using gemini-1.5-flash:

1. 24 combinations with 5 factors of 2+ levels (2 1 Shots Examples, 2 Domain Models to generate, 3 Levels of Task Decomposition, and 2 levels of Completeness Ratio for the Partial Solution Model).
2. 24 combinations with 4 factors of 3+ levels (2 1 Shots Examples, 3 Domain Models to generate, and 4 levels of Task Decomposition).

The first dataset is to find the significant effects of the all settings and their iteration. The second dataset is focused on the effects of the Partial Solution Model Completeness Ratio on the Domain Model Elements and if there are significant differences.

Table 2 shows the number of classes, attributes and relations for each Domain Solution Model as well as their 3 respective Partial Solution Model.

Table 3 shares the obtained average completeness ratio for each level of Partial Solution Model and for each element type inside it. The ratio tends to be pretty close on the classes but is a lot higher for the attributes and the relations. This is because the root classes passed tended to have the main part of the semantic of the solution model.

4.2. Research Questions

This sections details the findings from the datasets obtained through Linear Models in R studio. The notebook and the visualizations are available on the GitHub (12). This paper focuses on analyzing two settings:

1. Research Question 1: How does Task Decomposition influence the structure and quality of the generated domain model?
2. Research Question 2: How does the ratio of completeness of the Partial Solution Model affects the quality of the generated model?

4.2.1. RQ1: How does Task Decomposition influence the structure and quality of the generated domain model?

Table 4 shows the estimated averages of each task decomposition settings. The best F1 Score for the class is found with Auto with 0.77% F1 Score. However, the best attribute and relation F1 Score found are with the "None" control with 0.67 and 0.61.

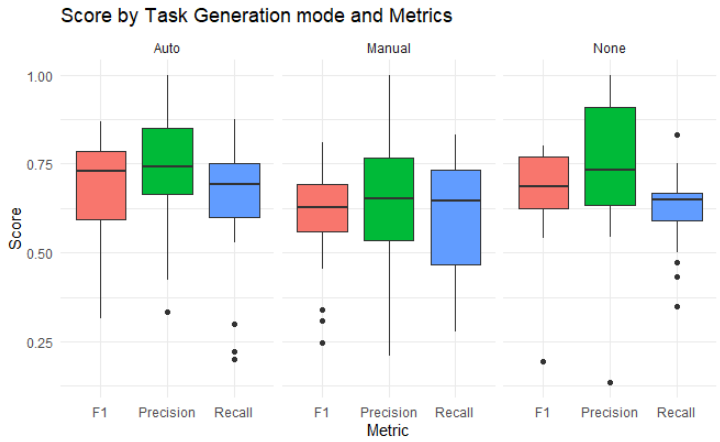

Figure 5: Precision, Recall and F1 Score for each Task Decomposition setting

Table 4: Estimated mean scores for each Task Decomposition setting. (emmean/SE, best F1 scores in bold)

	Class			Attribute			Relationship		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Auto	0.86 ± 0.04	0.73 ± 0.03	$\mathbf{0.77 \pm 0.02}$	0.63 ± 0.04	0.68 ± 0.02	0.64 ± 0.02	0.72 ± 0.06	0.50 ± 0.04	0.57 ± 0.04
Manual	0.78 ± 0.04	0.67 ± 0.03	0.70 ± 0.02	0.61 ± 0.04	0.60 ± 0.02	0.59 ± 0.02	0.49 ± 0.06	0.51 ± 0.04	0.49 ± 0.04
None	0.83 ± 0.04	0.67 ± 0.03	0.72 ± 0.02	0.79 ± 0.04	0.59 ± 0.02	$\mathbf{0.67 \pm 0.02}$	0.64 ± 0.06	0.60 ± 0.04	$\mathbf{0.61 \pm 0.04}$

Table 5 shows the contrast of estimated means between Auto and Manual Task Decomposition with the "None" control and their p-values(bold if significant). Nothing is significant and except Auto generated Classes all of them show that the generated were found to have a lower F1 Score when using the Task Decomposition options.

Table 5: F1 Score contrasts of the Task Decomposition methods with the control. (contrast/p-value, significant in bold)

	Class	Attribute	Relation
Auto	0.0517 / 0.2477	-0.0214 / 0.7853	-0.0373 / 0.8271
Manual	-0.0128 / 0.9116	-0.0709 / 0.0989	-0.1222 / 0.1615

To search for a potential interaction between the Task Decomposition mode and the Partial Solution Model, Table 6 shows the Task Decomposition settings estimated means whether there was a 33% Corresctness ratio PSM

or not. The highest F1 Score results where all obtained with the PSM: Class: 0.79, Attribute: 0.74 and Relation: 0.75. This is a high upgrad for the Attribute and the Relations increasing of 0.19 and 0.36 their respective F1 Score.

Table 6: Estimated mean scores for each Task Decomposition setting whether there is an input partial solution model or not. (emmean/SE, best F1 scores in bold)

		Class			Attribute			Relationship		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
No PSM	Auto	0.87 ± 0.06	0.68 ± 0.04	0.74 ± 0.03	0.51 ± 0.05	0.63 ± 0.03	0.55 ± 0.03	0.61 ± 0.08	0.31 ± 0.06	0.39 ± 0.06
	Manual	0.75 ± 0.06	0.59 ± 0.04	0.63 ± 0.03	0.61 ± 0.05	0.52 ± 0.03	0.54 ± 0.03	0.32 ± 0.08	0.38 ± 0.06	0.34 ± 0.06
	None	0.82 ± 0.06	0.65 ± 0.04	0.69 ± 0.03	0.78 ± 0.05	0.61 ± 0.03	0.68 ± 0.03	0.55 ± 0.08	0.55 ± 0.06	0.53 ± 0.06
33%PSM	Auto	0.84 ± 0.06	0.77 ± 0.04	$\mathbf{0.79 \pm 0.03}$	0.75 ± 0.05	0.74 ± 0.03	$\mathbf{0.74 \pm 0.03}$	0.83 ± 0.08	0.69 ± 0.06	$\mathbf{0.75 \pm 0.06}$
	Manual	0.81 ± 0.06	0.75 ± 0.04	0.78 ± 0.03	0.62 ± 0.05	0.68 ± 0.03	0.65 ± 0.03	0.66 ± 0.08	0.64 ± 0.06	0.64 ± 0.06
	None	0.84 ± 0.06	0.70 ± 0.04	0.75 ± 0.03	0.80 ± 0.05	0.56 ± 0.03	0.65 ± 0.03	0.73 ± 0.08	0.66 ± 0.06	0.69 ± 0.06

Table 7 shows the contrasts of estimated means for each Task Decomposition setting F1 Score with the control with and without a PSM and the p-value to know if the number is significant. Attribute generation is significantly worthed of (Auto:) -0.125 and (Manual:) -0.134 without a PSM.

Table 7: F1 Score contrasts of the Task Decomposition methods with the control whether there is an input partial solution model or not. (contrast/p-value, significant in bold)

		Class	Attribute	Recall
No PSM	Auto	0.0597 / 0.3847	**-0.12522 / 0.0347**	-0.1404 / 0.2858
	Manual	-0.0534 / 0.4609	**-0.13481 / 0.0228**	-0.1955 / 0.1037
33%PSM	Auto	0.0438 / 0.5882	0.08251 / 0.1936	0.0658 / 0.7456
	Manual	0.0279 / 0.8031	-0.00692 / 0.9872	-0.0488 / 0.8500

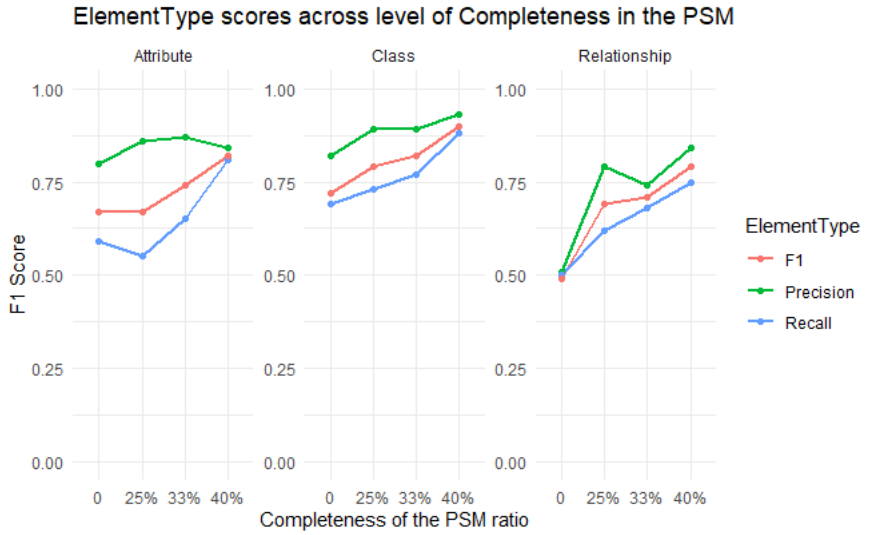

Figure 6: Classes, Attributes and Relations average metrics across the different levels of PSM

Answer to RQ1: Based on the test conducted, and on the estimated means and p-values of F1 Score, Task Decomposition does not consistently increase the quality of generated domain models. Linear Models, including estimated means and p-values for F1 scores, indicated no substantial evidence that Task Decomposition improves model quality. Furthermore, using Task Decomposition methods (Auto and Manual) without a Partial Solution Model (PSM) resulted in much worse performance in certain areas [7], such as attribute generation. Specifically, the F1 scores for characteristics decreased by 0.125 ($p = 0.0347$) and 0.135 ($p = 0.0228$) for the Auto and Manual settings, respectively, when compared to the control. However, no interaction impact was found between Task Decomposition and the presence of a PSM, suggesting that these methods operate independently of one another.

4.2.2. RQ2: How does the ratio of completeness of the Partial Solution Model affects the quality of the generated model?

Table 8 shows the metrics scores for the 3 level of Partial Solution Model. We see that all the best F1 score are with completeness ratio of 40%: Class 0.90, Attribute 0.82 and Relation 0.79.

Table 8: Estimated mean scores for each Partial Solution Model Completeness ratio. (emmean/SE, best F1 scores in bold)

	Class			Attribute			Relationship		
Completeness Ratio	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
40%	0.93 ± 0.04	0.88 ± 0.03	$\mathbf{0.90 \pm 0.03}$	0.84 ± 0.04	0.81 ± 0.05	$\mathbf{0.82 \pm 0.04}$	0.84 ± 0.06	0.75 ± 0.04	$\mathbf{0.79 \pm 0.05}$
33%	0.89 ± 0.04	0.77 ± 0.03	0.82 ± 0.03	0.87 ± 0.04	0.65 ± 0.05	0.74 ± 0.04	0.74 ± 0.06	0.68 ± 0.04	0.71 ± 0.05
25%	0.89 ± 0.04	0.73 ± 0.03	0.79 ± 0.03	0.86 ± 0.04	0.55 ± 0.05	0.67 ± 0.04	0.79 ± 0.06	0.62 ± 0.04	0.69 ± 0.05
0	0.82 ± 0.04	0.69 ± 0.03	0.72 ± 0.03	0.80 ± 0.04	0.59 ± 0.05	0.67 ± 0.04	0.51 ± 0.06	0.50 ± 0.04	0.49 ± 0.05

Table 9 shows that the classes have a significantly higher F1 score with a 40% PSM (0.1829/0.0018) and relations F1 scores are always significantly higher with a PSM.

Table 9: F1 Score contrasts between each Partial Solution Model Completeness ratio with the control. (contrast/p-value, significant in bold)

PSM	Class	Attribute	Relation
40%	**0.1829 / 0.0018**	0.15498 / 0.0530	**0.297 / 0.0018**
33%	0.0994 / 0.0928	0.07727 / 0.4575	**0.218 / 0.0191**
25%	0.0722 / 0.2716	0.00485 / 0.9975	**0.199 / 0.0018**

Answer to RQ2: The level of completeness ratio of the input Partial Solution Model's (PSM) was discovered to have a substantial influence on the quality of the generated domain models. Models with a 40% PSM completeness ratio had the greatest F1 ratings for all elements: Classes (0.90), Attributes (0.82), and Relations (0.79). Estimated Means and F1 Score contrasts of the fitted Linear Model indicated that the 40% completion ratio had a substantial impact on class creation, with an F1 score increase of 0.1829 ($p = 0.0018$) compared to the control group. Relation generation also showed significant gains, beginning with a 0.20 rise in F1 scores at 25% completeness ($p = 0.0018$) and progressing to a 0.30 increase at 40% completion. However, regardless of the completeness ratio, attribute creation did not improve significantly.

5. Discussion

5.1. Generation of Domain Models

Comparison to Related Work: Previous researches were done on the generation of domain models from a textual domain description like [(4), (7)]. This paper reused their datasets of Problem Description and solutions, their evaluation method.
Especially, (4) found that the generated suffered from a very low recall, that the Chain-of-Thought had negative unsignificant effect, and that Few shots was positively significant, but that there wasn't a significant difference between 1 and 2 shots.
(7) had very positive results by using a long iterative process they created called Multi-step Iterative Generator (MIG) where they focused on having the model recognized all the nouns for classes and attributes, use abstract classes and inheritance to reproduce common techniques in the domain modeling community, and generate relationships at the end.

5.2. Task Decomposition

Summary of key findings: This paper used an heuristic evaluation method to compute the F1 Score of generated domain models compared to a solution model from a domain-expert.
It was found that Task Decomposition setting alone tends to make the model elements F1 score worst than before. Also no interaction was found between this setting and the level of completeness of a potential input Partial Solution Model.

Comparison to Related Work: This shows that separating a complex textual description in subtasks, either built manually or automatically, performed worst in this scenario of domain modeling.
In a previous research (6), Cui et al. used task decomposition for a task of Named Entity Recognition with BART to create subtasks as follows:

1. Find all the possible compositions of segments of a sentence, ie "Pierre eats", "eats an apple", ...
2. Find all the entities and types for the compositions.
3. Evaluate the entities and their types.
4. Select the highest probability entity and type.

This allowed for better performances with limited ressources, better identification of the entities and also a better generalisation between domains.
Moreover, another research from Princeton University and Google Research (13) created an iterative prompt method called *ReAct* (Reasoning and Action) in which the model receives a complex tasks, decompose into subtasks and iteratively do:

1. Do a task (*Action*),
2. Observe the result (*Observation*),
3. Reason on the observation for next action (*Reason*), ...

This allows to reason on the next step after each subtasks making the process more robust.

Strength of the Approach: The different setting options of task decomposition tested for a wide range of decomposition.

- "Manual" delimitates the knowledge passed to the model at once, and gives control to the user to choose what and how much.
- "Auto" delimitates the knowledge passed to the model at once, leveraging the quantity of work asked to the user by letting an LLM define the tasks to do.

5.3. Partial Solution Model

Summary of key findings: It was found that 90% F1 score average was obtained for classes, 82% for attributes, and 79% for relations. Furthermore the F1 score contrasts were found significantly better for the classes with a 40% PSM, and for the relations with all 3 levels of PSM.

Comparison to Related Work: Previous researches [(9),(10)] have been done on creating domain modeling assistants with Deep Learning models, BERT and GNN, where they gave unfinished domain models and would get suggestion from the model on what to do next.

5.4. Limitations of the Approach

- Heuristic Evaluation: The evaluation of the models being heuristic is subjective and might not represent the real performance of the model for those settings.

- Limited dataset: The entire evaluation was done on gemini 1.5 flash, only on two 1-shot examples and on 2 to 3 Problem Domain. Also, didn't take in account temperature of the model.

- Disproportion of Completeness ratios: The completeness ratios varies from the intended ones and also between element types with an average of 53% of the solution attributes in the intended 40% PSM and only 42% of solution classes see table 3.

6. Conclusion

This paper describes the implementation and the evaluation of a modeling assistant system using the progress in Large Language models. The implemented model is fully automated and allow users to input a partial model they want to complete. It has been shown that adding a partial model containing 48% of the domain model elements is required to reach a 90% F1 score of classes, 82% for attributes and 79% for relations. However, using task decomposition settings didn't show a positive significant change.

Future Works: In the future, I would try to test on other models like the GPTs or the new gemini 2.0 model. As well as try other prompting method like ReAct which sounds very promising because it adds self reflexivity on the subtasks performed in the generation process which I think this implementation lacks.

References

[1] Eduardo Guerra, Maurício Aniche.
Achieving quality on software design through test-driven development
From Sotfware Quality Assurance, pp. 201-220 (2016).
DOI: `https://doi.org/10.1016/B978-0-12-802301-3.00009-0.`.

[2] Thomas Kühne.
Matters of (Meta-) Modeling.
Softw Syst Model 5, pp. 369–385 (2006).
DOI: `https://doi.org/10.1007/s10270-006-0017-9`.

[3] OpenAI.
Introducing ChatGPT

November 30th, 2022.
DOI: https://openai.com/index/chatgpt/.

[4] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Mussbacher, and Dániel Varró.
Automated Domain Modeling with Large Language Models: A Comparative Study.
In ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS 2023), October 1-6, 2023, Västerås, Sweden. IEEE, pp. 162-172, 2023.
DOI: https://doi.org/10.1109/MODELS58315.2023.00037.

[5] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, Graham Neubig.
Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing.
DOI: https://doi.org/10.48550/arXiv.2107.13586.

[6] Leyang Cui, Yu Wu, Jian Liu, Sen Yang, Yue Zhang.
Template-Based Named Entity Recognition Using BART
DOI: https://doi.org/10.48550/arXiv.2106.01760.

[7] Yujing Yang, Boqi Chen, Kua Chen, Gunter Mussbacher, and Dániel Varró.
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.
DOI: https://doi.org/10.1145/3652620.3687807.

[8] B. Chen et al.
On the Use of GPT-4 for Creating Goal Models: An Exploratory Study.
In IEEE 31st International Requirements Engineering Conference Workshops (REW), Hannover, Germany, pp. 262-271, 2023.
DOI: https://doi.org/10.1109/REW57809.2023.00052.

[9] Weyssow, M., Sahraoui, H. and Syriani, E.
Recommending metamodel concepts during modeling activities with pretrained language models.

Softw Syst Model 21,
pp. 1071–1089, 2022.
DOI: https://doi.org/10.1007/s10270-022-00975-5.

[10] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, Phuong T. Nguyen.
A GNN-based Recommender System to Assist the Specification of Metamodels and Models.
2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), Fukuoka, Japan, 2021, pp. 70-81
DOI: https://doi.org/10.1109/MODELS50736.2021.00016.

[11] Eugene Syriani, Robert Bill, Manuel Wimmer.
Domain-Specific Model Distance Measures.
Journal of Object Technology, Volume 18, no. 3 (July 2019), pp. 3:1-19
DOI: https://doi.org/doi:10.5381/jot.2019.18.3.a3.

[12] GitHub repository available at: https://github.com/elnukakujo/OpenAI_Assistant_M2_Gen.git

[13] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao.
ReAct: Synergizing Reasoning and Acting in Language Models.
2023
DOI: https://doi.org/10.48550/arXiv.2210.03629.