

Université de Montréal

**Few-Shot Prompt Learning for Automating Model
Completion**

par

Meriem Ben Chaaben

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maîtrise sciences (M.Sc.)
en Intelligence Artificielle

August 31, 2023

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Few-Shot Prompt Learning for Automating Model Completion

présenté par

Meriem Ben Chaaben

a été évalué par un jury composé des personnes suivantes :

Eugene Syriani

(président-rapporteur)

Houari Sahraoui

(directeur de recherche)

Bang Liu

(membre du jury)

Sommaire

Les modélisateurs rencontrent souvent des défis ou des difficultés lorsqu'il s'agit de concevoir un modèle logiciel particulier. Dans cette thèse, nous avons exploré différentes voies et examiné différentes approches pour résoudre cette problématique. Nous proposons enfin une approche simple mais novatrice qui améliore la complétion des activités de modélisation de domaines. Cette approche exploite la puissance des modèles de langage de grande taille en utilisant l'apprentissage par seulement quelques exemples, éliminant ainsi la nécessité d'un apprentissage profond ou d'un ajustement fin (fine tuning) sur des ensembles de données rares dans ce domaine.

L'un des points forts de notre approche est sa polyvalence, car elle peut s'intégrer facilement à de nombreuses activités de modélisation, fournissant un aide précieux et des recommandations aux modélisateurs.

De plus, nous avons mené une étude utilisateur pour évaluer l'utilité de cette méthode et la valeur de l'assistance en modélisation; nous avons cherché à savoir si l'effort investi dans l'assistance en modélisation vaut la peine en recueillant les commentaires des concepteurs de modèles logiciels.

Mots-clés: Ingénierie basée sur les modèles, modèles de langage, apprentissage par quelques exemples, apprentissage basé sur des instructions, modélisation de domaines, complétion de modèles, étude utilisateur

Summary

Modelers often encounter challenges or difficulties when it comes to designing a particular software model. Throughout this thesis, we have explored various paths and examined different approaches to address this issue. We finally propose a simple yet novel approach enhancing completion in domain modeling activities. This approach leverages the power of large language models by utilizing few-shot prompt learning, eliminating the need for extensive training or fine-tuning on scarce datasets in this field. One of the notable strengths of our approach lies in its versatility, as it can be seamlessly integrated into various modeling activities, providing valuable support and recommendations to software modelers. Additionally, we conducted a user study to evaluate the usefulness of this approach and determine the value of providing assistance in modeling; we aimed to determine if the effort invested in modeling assistance is worthwhile by gathering feedback from software modelers.

Keywords: Model-Driven Engineering, Language Models, Few-Shot Learning, Prompt Learning, Domain Modeling, Model Completion, User Study

Contents

Sommaire	5
Summary	7
List of tables	13
List of figures	15
Acronyms & Abbreviations	17
Dedications	19
Acknowledgements	21
Chapter 1. Introduction	23
1.1. Research context.....	23
1.2. Problem statement.....	24
1.3. Main contributions.....	25
1.4. Thesis structure	26
Chapter 2. Background and Related Work	27
2.1. Background.....	27
2.1.1. MDE	27
2.1.1.1. MDE artefacts and transformations	27
2.1.1.2. General propose modeling formalisms; Graphical models	29
2.1.1.3. Intelligent modeling assistance	30
2.1.2. Techniques in Natural Language Processing	31
2.1.2.1. Large language models.....	31
2.1.2.2. Prompt engineering.....	33
2.1.2.3. Few shot learning.....	34
2.2. Related work	35

2.2.1.	Leveraging domain models data for model completion (1)	37
2.2.2.	Utilizing code data for model recommendation (2)	38
2.2.3.	Extracting domain models from natural language sources (3)	38
2.3.	Conclusion	39
Chapter 3. Exploring the Usage of Code Data for Model Completion		41
	Introduction	41
3.1.	Context and Proposed Approach	41
3.1.1.	Reverse Engineering and Data Extraction	42
3.1.2.	Preprocessing and Learning Phase	44
3.1.2.1.	Data Cleaning	44
3.1.2.2.	Data Encoding	44
3.1.2.3.	Learning Phase	45
3.2.	Evaluation and Discussion	46
3.2.1.	Setup	46
3.2.2.	Results	47
3.2.2.1.	Suggesting concepts:	47
3.2.2.2.	Classifying association types:	47
3.3.	Discussion	48
3.4.	Conclusion	48
Chapter 4. Few-Shot Prompt Learning for Automating Model Completion		51
	Introduction	51
4.1.	Context and Motivations	51
4.2.	Approach	52
4.2.1.	Static diagram completion: Class diagrams	53
4.2.1.1.	Class Names suggestions	54
4.2.1.2.	Attributes suggestion	56
4.2.1.3.	Associations names and types suggestion	57
4.2.2.	Dynamic diagram completion: Activity diagrams	59
4.3.	Preliminary evaluation	60
4.3.1.	Setup	61

4.3.1.1.	Class names suggestions	62
4.3.1.2.	Attributes suggestion	62
4.3.1.3.	Association names suggestion	62
4.3.2.	Results	62
4.3.2.1.	Class names suggestion	62
4.3.2.2.	Attributes suggestion	63
4.3.2.3.	Association names suggestion	63
4.4.	Conclusion	64
Chapter 5.	On the usefulness of Prompt Based Modeling Assistance.....	65
Introduction		65
5.1.	Tool support	65
5.1.1.	Developing a DSM Graphical Editor for class diagrams	66
5.1.2.	Recommendation modes	66
5.1.3.	Implementation details	68
5.2.	Evaluating the usefulness of Modeling Assistance	69
5.2.1.	Aim and scope	69
5.2.2.	Research questions	70
5.2.3.	Experimental setup	70
5.2.3.1.	Tasks: Recommendation alternatives	71
5.2.3.2.	Modeled domains	71
5.2.3.3.	Variables	71
5.3.	Quantitative Analysis	73
5.3.1.	Time to complete a task	73
5.3.2.	Diversity of solutions	75
5.3.3.	Contribution of suggestions	77
5.3.3.1.	Acceptance Rate (concepts)	77
5.3.3.2.	Assessment Rate (concepts)	78
5.4.	Qualitative Analysis	79
5.4.1.	Results during experiments	79
5.4.2.	Effectiveness of the recommendations	80
5.4.3.	Impact of the tool	82
5.4.4.	Participants feedback	83

5.5. Threats to validity	84
5.6. Conclusion	85
Chapter 6. Conclusion and future work	87
6.1. Summary	87
6.2. Future work	87
Références bibliographiques	89
Appendix A	93
.1. Video tutorial	93
.2. Forms	93
.3. Tasks descriptions	96

List of tables

3.1	Association type prediction evaluation	48
4.1	Semantic Mapping for Activity Diagrams	59
4.2	Results evaluating class names prediction	63
5.1	The goal of the user study.....	70
5.2	Experimental design	71
5.3	Results evaluating the time spent for each mode	73
5.4	Pairwise comparison of treatment- Time To complete the task.....	74
5.5	Results evaluating the diversity between obtained models cross different modes..	75
5.6	Pairwise comparison of treatment- Diversity	76
5.7	Results evaluating the Acceptance rate for each mode	77
5.8	Pairwise comparison of treatment- Acceptance Rate.....	77
5.9	Results evaluating the Assessment rate for each mode	78
5.10	Pairwise comparison of treatment- Assesment Rate.....	78

List of figures

2.1	Overview of the artifacts relation with the metamodel [21]	28
2.2	A class diagram example	29
2.3	Illustration of activity diagrams	30
2.4	Architecture of large language models, adapted from [48]	32
2.5	Explanatory example of in-context learning	35
2.6	Overview about the approaches in the literature	37
3.1	Illustration of the process used to do model completion starting from Java code .	42
3.2	Illustration of the process used to encode concepts of a Java repository.	43
3.3	Illustration of the process used to identify the association type between two concepts.....	45
4.1	Overview of the approach: using few shot learning to automate modeling completion -one iteration-.....	53
4.2	Example of domain model: Bank class diagram - Partial model	53
4.3	Prompt and generated text for class names and association prediction to complete the Class Diagram of Fig. 4.6	54
4.4	Example of domain model: Bank class diagram - updated model with suggested concepts.....	55
4.5	Prompt and generated text for attribute completion.....	56
4.6	Example of domain model: Bank class diagram - updated model with suggested attributes.....	57
4.7	Prompt and generated text for association name completion.....	57
4.8	Prompt and generated text for association type completion.....	58
4.9	Example of domain model: Bank class diagram - updated model with suggested type and names of associations.....	58
4.10	Example of activity diagram: Online Shopping	60

4.11	Prompt and generated text for the Activity Diagram of Fig. 4.10.....	60
4.12	Results evaluating our approach	64
4.13	Results in terms of Recall@20 in the work of Weysow et al. [50]	64
4.14	A comparative display of findings in two Independent works	64
5.1	Ecore metamodel of Class diagram editor	67
5.2	Implementation Snapshot: On request Mode	68
5.3	Implementation Snapshot: Automatic Mode	68
5.4	Answers to question: During the experiment, I felt.....	80
5.5	Answers to question: I found the recommendations.....	81
5.6	Answers to question: I found the tool....	83

Acronyms & Abbreviations

MDE	Model-Driven Engineering
UML	Unified Modeling Language
NLP	Natural language processing
NL	Natural language
LLM	Large Language Models
GPT	Generative Pre-trained Transformer
AI	Artificial Intelligence

Dedications

Thank you dear God for all your blessings.

My humble effort I dedicate to:

*my loving mother for all the sacrifices you make each day and for giving me all the support
and love I need.*

*you are my **joy of living**;*

my caring father for being my first teacher and encouraging me to believe in myself.

*you are my **hero**;*

my creative sister, for inspiring me and giving me hope.

*you are my **treasure**;*

my smart brother, for filling my soul with joy.

*you are my **pride**;*

my friends, for being there for me and cheering me up

my entire extended family, for being supportive.

May god keep us always together

and to all my teachers, for their guidance and dedication.

Thank you...

Gratitude is riches

Acknowledgements

I express my deep sense of gratitude to anyone who helped me and gave me support during this research work.

I express my deepest gratitude to both institutions; the Tunisian Ministry of Higher Education and Scientific Research and the University of Montreal for co-funding this research work through various excellent scholarships. Their support has been instrumental in shaping my research career and has certainly had a lasting impact on my academic journey.

I acknowledge with thanks the timely guidance which I received from my academic supervisor Professor. Houari Sahraoui. I am sincerely thankful for his continuous support, encouragement, and advice throughout this research journey.

I would like to thank my fellow lab members for their assistance, insightful discussions, and constructive feedback. I am grateful for the stimulating environment they have created, which fostered creativity and innovation.

Finally, I extend my sincere thanks to all the members of the jury for their time, expertise, and dedication to evaluating my research work.

Chapter 1

Introduction

1.1. Research context

Model-driven engineering (MDE) is a frequently used development technique that helps manage the complexity of the problem at hand and increases the efficiency and effectiveness of software development through the power of abstraction [11, 42]. Models, through the right domain-specific views, close the gap between the problem at hand and the level of human reasoning. However, when designers ignore the domain, they may confront several challenges that can make completing design models considerably more difficult. One of the significant challenges is time constraints. Without a clear understanding of the domain, designers may spend a lot of time trying to comprehend the requirements and design models, causing delays in the software development cycle. Furthermore, the complexity of the task can increase as the designer may not be able to identify all the necessary elements for completing the design model. As a result, designers may ignore critical aspects of the design, leading to errors and omissions that can impact the quality of the final software application [25]. These challenges have led to an increased demand for new tools and techniques that can assist experts in this task.

On the other hand, artificial intelligence (AI) has recently seen a remarkable evolution. One of the most prosperous fields nowadays is artificial intelligence, notably its subfields machine learning and deep learning. As a result of its formidable ability to mimic human intelligence and create intelligent systems, AI has become a highly sought-after and widely employed technology in a variety of fields, and one of the most interesting fields where we could leverage the power of AI is software engineering [20]. In recent years, so many AI-based tools have been designed and developed as well as several research have been conducted in order to utilize it to assist software specialists with their duties, optimize pipelines, and enhance software quality.

And similar to text editing, programming, and testing [36, 47], the design phase can benefit from high-quality completion mechanisms and recommendation systems that might shorten the time required to carry out a task and improve its quality.

To summarize, MDE simplifies software development through abstraction, but challenges like time constraints and incomplete domain understanding, hinder the design process. This demands new tools. AI advancements offer opportunities to leverage intelligent systems in software engineering, including design, by improving completion mechanisms and recommendations.

1.2. Problem statement

Although recent developments in deep learning-based language models (LMs) open a world of possibilities to automate and assist software specialists in software development and maintenance tasks [47], these opportunities are limited when dealing with the early software development phase, i.e modeling and design.

Generating high-quality recommendations can only be achieved by analyzing a vast number of models and learning concepts and associations from a training set. Unfortunately, as modeling languages tend to be highly specialized to the domain at hand, training data is scarce and does not allow for training or fine-tuning a high-quality recommender logic.

Existing approaches in the current state of the art have taken three primary perspectives for design assistance, each drawing on different sources of information and techniques;

- (1) Considering the modeling data itself as a starting point [33, 50]: This perspective encourages employing design techniques only and doesn't include other implementations phases. Techniques like constraint logic programming, and similarity detection can be utilized to further improve the accuracy of the design completion process [44].
- (2) Leveraging data retrieved from code: This perspective involves utilizing code analysis techniques in the context of object-oriented programming (OOP) to extract relevant data and patterns from existing software projects and learn domain concepts and their relationships from identifiers [16]. Analyzing the codebase provides valuable insights and information that can assist in the design phase.
- (3) Leveraging knowledge from various textual resources: This perspective focuses on retrieving data from specifications, documentation, requirements, design artifacts, and general knowledge from diverse domains. By employing natural language processing (NLP) techniques, it enables the extraction of relevant information and the inference of patterns from textual analysis and general knowledge, thereby offering design assistance based on a broader understanding of the problem domain.

While these perspectives offer distinct approaches, they have traditionally relied on either *training* or *fine-tuning* models based on the available data. In summary, to effectively address

the limitations and complexities of the early software development phase, there is a need for a comprehensive and effective solution.

Problem 1: How can we effectively mitigate the data scarcity to perform modeling completion task?

Data quality is of utmost importance in modeling completion tasks, especially considering the utilization of AI techniques. The accuracy and reliability of AI models heavily depend on the quality of the training data. In modeling completion, where diverse formalisms are used, having perfect and clean data is crucial. Each formalism requires specific data representations, tailored to the employed AI techniques. Exploring approaches among the previously mentioned ones to mitigate data scarcity is essential for effective modeling completion.

Problem 2: To what extent is assistance during the modeling phase effective and useful?

The purpose of this study is to assess the usability and the correctness of the chosen methodologies for aiding in the modeling phase. We evaluate the degree to which modeling assistance enhances the results of the design task by using a variety of appropriate approaches and metrics. The evaluation process will also involve determining if the suggested completions are accurate and whether they meet the desired modeling goals.

1.3. Main contributions

In our work, we will initially explore the option of leveraging data retrieved from code analysis techniques in the early software development phase. However, we acknowledge that the extracted code data needs to fit perfectly into each design task, as it may not be easily generalizable. This limitation prompts us to propose an alternative approach within the perspective of utilizing NLP techniques. By incorporating NLP techniques, we aim to overcome the challenge of data generalizability by encoding knowledge and specifications from various textual resources from diverse domains. By incorporating these insights into the design assistance system, we aim to enhance the quality and effectiveness of design recommendations in the early software development phase.

- Exploration of two different perspectives to mitigate the data scarcity and perform modeling completion task across different chapters:
 - (1) Presenting a code-based approach for model completion and evaluating its effectiveness and results. (Chapter 3)
 - (2) Introducing and evaluating a paradigm shift in model completion that utilizes NLP techniques, offering greater generalizability and enhanced efficiency. (Chapter 4)
- Development of a tool to support users in designing domain models using Model-Driven Engineering (MDE) artifacts. (Chapter 5)

- Evaluation of the usability of a model completion software powered by Large Language Models (LLMs). This evaluation assesses the user-friendliness, effectiveness, and efficiency of the model completion tool, providing insights into its practicality and potential for improving the model completion process. (Chapter 5)

1.4. Thesis structure

The remainder of this thesis is organized as follows.

Chapter 2 provides both background information and literature review on previous related works that are relevant to the main contributions of this thesis.

Chapter 3 focuses on a specific perspective chosen for exploration to answer the first problem. We delve into the utilization of Code Data to extract knowledge that can be employed for model assistance.

In Chapter 4, we shift our attention to a more promising approach that involves the utilization of already encoded general knowledge. We explore the application of Few-Shot Prompt Learning techniques for automating model completion and we present a preliminary evaluation of its correctness.

In Chapter 5, we evaluate the usefulness of the proposed approach. The evaluation encompasses various metrics and techniques to assess the performance and impact of the modeling assistance technique.

Finally, we conclude our thesis in chapter 6 by summarizing our contributions, underlining their limitations, and outlining future research directions.

Chapter 2

Background and Related Work

In this chapter, we provide all the background information required to introduce our work. Subsequently, we review the literature with regard to the main themes of this research work.

2.1. Background

In the coming sections, we build upon the principles of Model-Driven Engineering (MDE) and exploit the capabilities of Large Language Models (LLMs) to address the challenges and explore novel approaches in our research domain.

2.1.1. MDE

MDE stands for Model-Driven Engineering [42], which is a software engineering approach that emphasizes the use of models and model transformations to design, develop, and maintain software systems. MDE provides a systematic and structured approach to software development, enabling better communication, productivity, and maintainability throughout the software development life-cycle. It has already gained significant traction and is being widely used in various industries [7, 30]. By using models as a primary artifact, MDE allows developers to focus on the essential aspects of a system’s design and its core functionality rather than getting lost in low-level implementation details. This may result in better software quality and maintainability thus enhanced productivity.

2.1.1.1. MDE artefacts and transformations.

MDE involves the use of **models** as the primary assets in software development, from which other artefacts —like code or other models— may be derived in an automated way. Models conform to a **meta-model**, which defines the modelling language syntax and determines the set of models that are valid. Meta-models comprise a structural diagram plus additional constraints formulating restrictions that cannot be expressed diagrammatically. The structural diagram is defined as a class diagram, frequently

using standards like the Meta-Object Facility (MOF)¹ and implementations like the Eclipse Modeling Framework (EMF) and Ecore [46].

Additionally, MDE solutions may also include **model transformations**, which are essential in software development because they enable model conversion and manipulation. As illustrated in figure 2.1, these transformations permit the creation of new models, code, and other artifacts from existing models. They are essential for automating time-consuming and error-prone processes, boosting productivity, and preserving consistency throughout the development process.

Particularly, in handling data, MDE transformations are essential for transforming data models into appropriate formats for future use, enabling efficient data processing, analysis, and integration within software systems. Model transformations can be defined using specialized transformation languages such as ATL [43], general-purpose languages such as Java, or technologies such as XSLT [28]. These transformations can produce textual artifacts such as code, configuration files, and documentation from models.

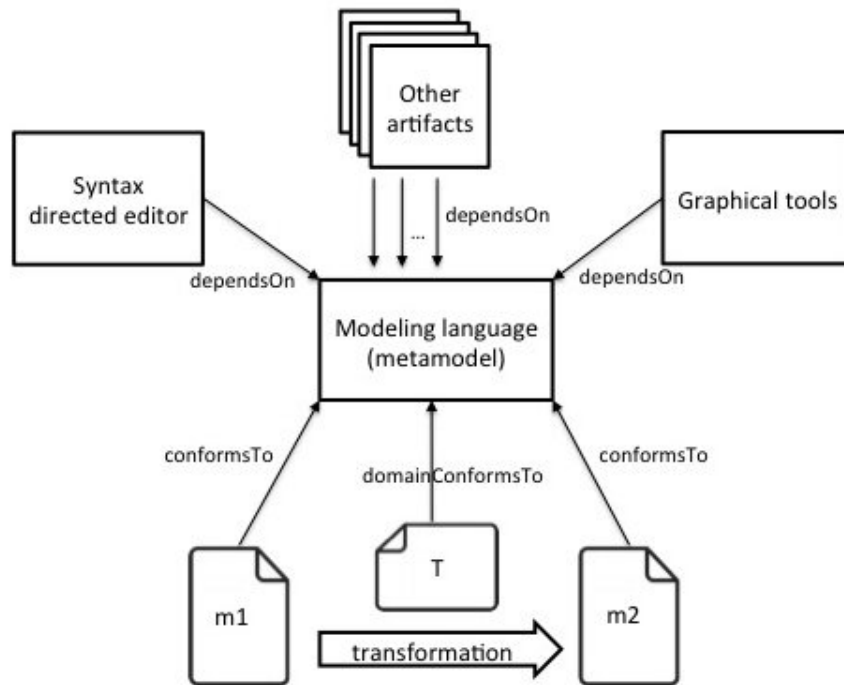


Fig. 2.1. Overview of the artifacts relation with the metamodel [21]

Modelling languages can be either *general-purpose*, such as UML, or *domain-specific*, developed for a specific domain. In this work the focus is on general purpose ones that are designed to be applicable across various domains and industries.

¹MOF 2.5.1. <https://www.omg.org/mof/> (2016)

2.1.1.2. General purpose modeling formalisms; Graphical models.

General purpose modeling formalisms are methods for representing and analyzing complex systems or processes. These formalisms provide a clear and concise method to represent the structure, behavior, and properties of a system.

Graphical models, on the other hand, are a type of general purpose modeling formalism that use graphs to represent the relationships between variables or components in a system. These models are especially useful in situations where there are numerous variables and intricate interactions to clarify complicated concepts and improve communication amongst multiple stakeholders.

The graphical nature of Unified Modelling Language (UML) [38] allows for a clear and concise representation of the system's structure, behavior, and interactions.

Diagrams used in UML may be categorised into various different categories. We distinguish first (1) **Static diagrams** that only describe the characteristics of a system or part of a system, like Class diagrams, Deployment diagrams and Package diagrams. Second, (2) we have **Dynamic diagrams** that describe the behaviour of the system such as Use Case diagrams, State Machine diagrams, and Activity diagrams.

In our research, we explore one modeling formalism from each group and thoroughly investigate the proposed approach within each of them. By considering multiple formalisms, we aim to broaden our understanding and assess the applicability of our approach across different contexts. This comprehensive analysis allows us to gain deeper insights into the strengths and limitations of our proposed method in diverse modeling frameworks.

The UML class diagram is one of the most prevalent types of graphical models in software engineering [10]. A class diagram is a form of UML diagram that describes the structure of a system by displaying the concepts in the system, their attributes, and their relationships as exemplified by the instance shown in Figure 2.2. It provides a clear perspective of the system's structure, making it easier to detect possible faults and improve the design. Developers, designers, and architects use them to convey ideas and collaborate on the development of software systems.

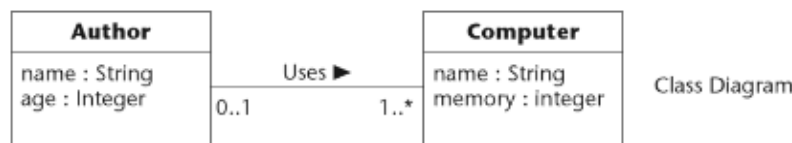


Fig. 2.2. A class diagram example

The UML activity diagram is another essential type of graphical model widely used in software engineering but unlike class diagrams, this diagram belongs to the dynamic diagrams family. In fact, it is used to illustrate the workflow portion of business processes or to describe the low level behaviour of software components. More precisely and as illustrated in 2.3, it showcases the sequence of actions and steps, decision points, concurrency, conditions, and control flows that occur during the execution of a process or algorithm.

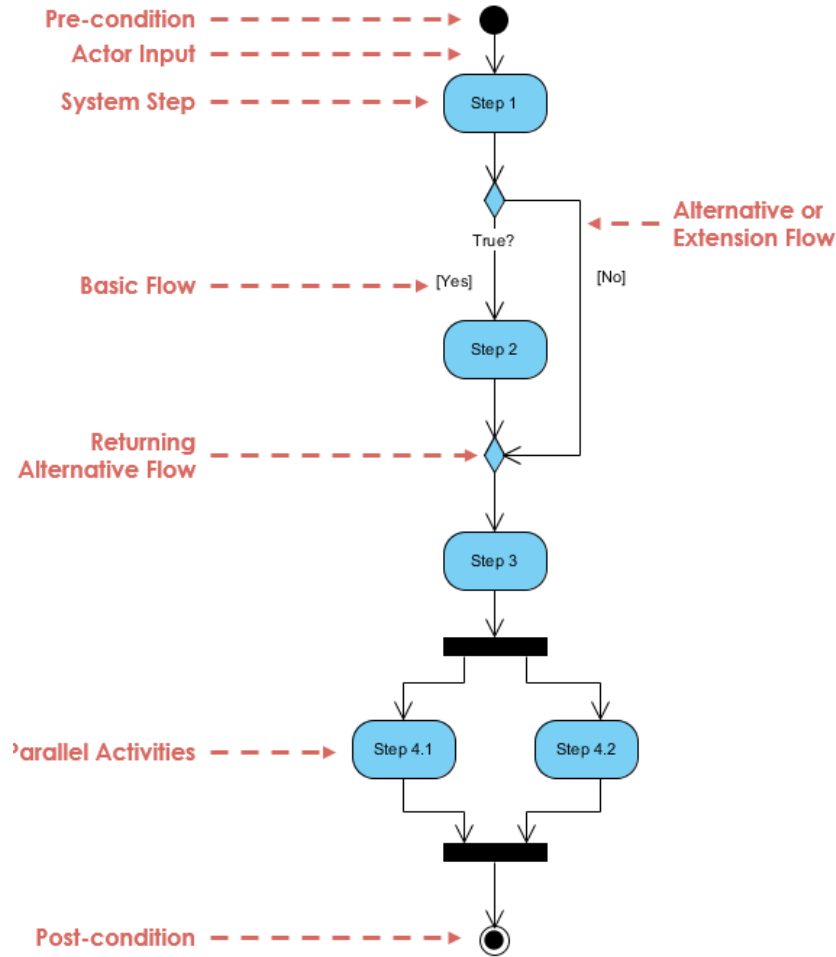


Fig. 2.3. Illustration of activity diagrams

2.1.1.3. Intelligent modeling assistance.

Creating accurate and useful models can be challenging, especially when dealing with large and complex systems. In fact, Understanding the intricacies of such systems and capturing them accurately in a model requires in-depth knowledge of the system's domain and architecture. Besides, mastering a modeling formalism often involves a learning curve, especially for complex languages like UML. overall, modelers need to invest both time and effort to gain proficiency in a given formalism.

Modeling assistance is an *iterative process* that involves collaboration between a human modeler and modeling tools or softwares. The goal of modeling assistance is to bridge the gap between the current state of a model (under construction M) and the intended or targeted model (completed model M') that aligns with the modeler's intentions for the ongoing modeling task. This assistance may come in the form of automated suggestions or feedback from modeling tools.

In a previous research work about intelligent modeling assistance [31], the authors came up with a set of metrics or properties that helps in evaluating an intelligent modeling assistance such as *quality of the obtained models, correctness and relevance of the suggested elements, and confidence in the assistance.*

In our work, we investigate both the efficiency or quality of the suggested elements and the usefulness of the provided assistance.

2.1.2. Techniques in Natural Language Processing

Natural language processing (NLP) has witnessed significant advancements in recent years, largely driven by the development of pre-trained language models.

2.1.2.1. Large language models.

Large language models (LLMs) such as GPT [34], GPT2 [35], GPT3 [13] have demonstrated remarkable capabilities in understanding, generating, and manipulating text.

The training process of the LLMs models typically entails self-supervised learning [35], in which the models learn to predict missing words or the next word in a sentence. This process enables the models develop a comprehensive understanding of language patterns, grammar, semantic relationships and textual contexts. These large models are founded on the dominant NLP architecture, the transformer architecture that is first presented in the work [48]. As illustrated in fig. 2.4, a transformer architecture is composed of multiple essential components within both its encoder and decoder, often referred to as layers. These layers, namely self-attention layers, operate by allowing each word in an input sequence to selectively focus on other words, thereby capturing intricate word dependencies essential for understanding context. Feed-forward layers contribute depth by applying transformations and non-linear functions, facilitating the extraction of complex patterns from the data. Multi-head attention extends this concept, utilizing multiple parallel attention mechanisms to capture diverse relationships within the sequence simultaneously. For the decoder, masked multi-head attention ensures the causality of predictions by

restricting attention to previous positions during text generation. Furthermore, normalization layers stabilize training by standardizing inputs, aiding in gradient flow and convergence acceleration. Working in harmony, these components synergistically empower transformers to comprehend language nuances and generate coherent text across various applications.

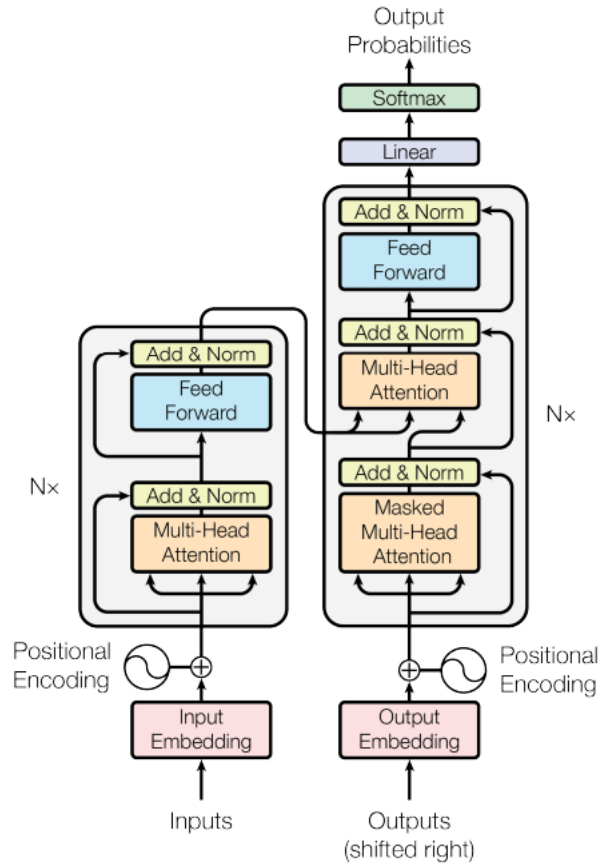


Fig. 2.4. Architecture of large language models, adapted from [48]

LLMs have demonstrated remarkable language generation abilities, allowing text completion, summarization, and even creative writing. These models can comprehend and generate coherent, context-appropriate text, which makes them valuable instruments for a variety of NLP applications [35]. They have been implemented in chatbots, virtual assistants, language translation systems, and content generation, among numerous other language-related applications.

It is essential to note that LLMs are typically trained using general language data and may lack domain-specific knowledge. However, end users or researchers should apply the appropriate approach to leverage the power of these models. With an additional training that uses domain-specific data, they can be fine-tuned or adapted to specific domains or tasks.

More specifically, and given the scarcity of domain-specific data in the field of modeling, an intriguing approach is to leverage the vast amount of encrypted learned general knowledge within LLMs. This eliminates the need for retraining the language models and instead focuses on establishing a semantic mapping that establishes equivalence between the formalism used in modeling and the textual format understood by LLMs [13]. Further details are presented in chapter 4.

This will lead us to introduce a technique that should assist in leveraging the power of these language models to accomplish particular objectives.

2.1.2.2. Prompt engineering.

Prompt template engineering refers to the process of developing a prompting function with the objective of achieving optimal performance on the downstream task. By carefully choosing suitable prompts, we can modify the behavior of the model in a way that the pre-trained language model alone can be employed to generate the desired output, often without requiring any additional training specific to a particular task [27]. The first aspect we focus on is the prompt shape. We distinguish two types:

- **Prefix Prompts:** In this type of prompt, a partial sentence or a specific context is provided as the initial input. The model is then expected to complete the sentence based on the given prefix. Prefix prompts can be useful for tasks where it is essential to generate coherent and context-appropriate outputs. By offering a relevant beginning point, these prompts direct the model’s generation process and aid in shaping its output.
- **Cloze Prompts:** Cloze prompts require us to provide a text with particular portions masked or eliminated. The model’s job is to fill in the blanks and generate the entire text. Cloze prompts are useful for tasks that require gaps to be filled, such as language comprehension, text completion (the case of our project), or responding to questions. Cloze prompts can successfully draw the model’s attention to the missing information and encourage correct and relevant predictions by deliberately hiding particular areas of the input.

Elaborating these prompts could be done *manually* which means they will be based on human introspection like the crafted prefix prompts that are presented in the work of Schick and Schütze [41]. Alternately, it could be accomplished *automatically* [45] through the use of algorithms that analyze existing data or examples related to the task and extract templates that encapsulate the desired input-output mappings. In our work, we carefully create the prompts manually, as illustrated in chapter 4. An important step in prompt engineering is including a task description in the prompt to help set the expectations for the model and provide it with a clear understanding

of the desired task or goal. The next step could be providing labeled examples to guide the model. This paradigm is called **Few shot learning**.

2.1.2.3. Few shot learning.

While fine-tuning has been a well-liked method for adapting to new tasks [26], it was previously considered challenging to fine-tune extremely large models, especially those with over 10 billion parameters. OpenAI have showcased the feasibility of this pursuit thanks to groundbreaking innovations in optimization techniques, substantial computational resources, distributed computing infrastructure, and data augmentation strategies. Expertise in hyperparameter tuning, data augmentation, and domain knowledge is essential, as is access to high-quality and diverse datasets. Overfitting challenges, diminishing returns from larger models, deployment complexities, and ongoing research efforts further contribute to the rarity of replicating such accomplishments in the machine learning field.

Brown et al. [13] propose using in-context learning as an alternative method for mastering new tasks. This technique consists in giving very few examples (few shots) on the kind of information we are looking for and then give a specific text to complete (which we will call *query*). LLMs can learn to perform a new task if they are given a few examples of what the task should look like, rather than trying to learn the task from scratch. This technique has proven its efficiency to various natural language processing tasks [27].

In the following, we present a simple example where we explain these concepts. If the goal is to use a LLM to generate the name of the capital of a given country, we need to prepare a prompt where we first provide the LM a description on how to reply to our queries, then we add relevant labeled samples such as two countries and their corresponding capitals (two shots). Finally, we give the country for which we want the LLM to provide its capital, i.e., autocomplete. Figure 2.5 gives an overview on how to use LLM with prompt learning applied to this example of countries and capitals. Note that apart from being an autocomplete engine, these language models are also pattern matching and pattern generation engines. This is why we need to provide not only information about the task to perform but also the pattern that we want them to replicate. In our example, each prediction unit is a token or span and not an entire text, as a result it is recommended to put the max tokens parameters to a low value. The generated text will try to fit the pattern in accordance with how the prompt is written. As our example shows in Figure 2.5, the generated text could include further information. For instance, `Japan => Tokyo` has also been generated, which is not of our interest. To deal with this behaviour, properly transforming

queries into prompts and the generated text into the appropriate result of the task is an essential part of the pipeline.

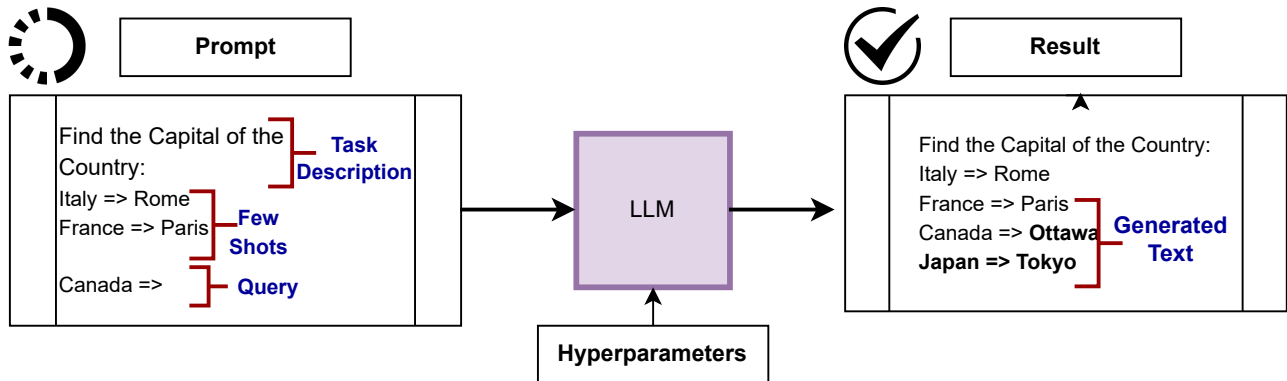


Fig. 2.5. Explanatory example of in-context learning

To summarize, without using gradient updates, the LM learns a new task solely through inference by conditioning on a concatenation of the training data as examples. The work [13] conducted experiments to evaluate the impact of natural language task descriptions and the amount of labeled examples on model performance. The results indicate that the addition of a natural language task description and increasing the number of labeled examples in the model’s context both contribute to improved performance.

The same study proved that few-shot learning also improves considerably with model size. Though the results in this scenario are particularly remarkable, the overall patterns with model size and quantity of in-context examples hold for the majority of tasks they analyze.

This approach enables the utilization of LLMS for modeling tasks in domains with limited data availability. It opens up opportunities for efficient and accurate model completion by exploiting the power of LLMS without the need for extensive retraining efforts or large amounts of domain-specific data.

2.2. Related work

Recent developments in deep learning-based language models (LMs) open a world of possibilities to automate and assist software specialists in software development and maintenance tasks within programming IDEs [36].

These opportunities are, however, limited when dealing with early software development phases such as analysis and design, although systems are becoming increasingly complex thus the field of modeling necessitates greater efforts to effectively address this inherent complexity.

When addressing this issue, two critical aspects need to be considered: the selection of appropriate AI models for prediction and completion tasks, and the availability and quality of the corpus or data used to train and feed these models.

Regarding AI models, there are two main approaches: using **pretrained** models that have been trained on large-scale datasets or **fine-tuning** them specifically for software engineering tasks. Pretrained models offer the advantage of leveraging existing knowledge and language understanding, while fine-tuning allows customization and adaptation to the specific needs of the software engineering domain.

However, regardless of the chosen approach, the availability and quality of the dataset play a critical role. Unfortunately, datasets in the software engineering domain are often scarce, and when available, they are not large enough to pre-train or fine-tune deep-learning models. For software modeling activities, several contributions were proposed to circumvent the lack of large datasets. The goal of these research contributions is to recommend domain concepts, their features, and relationships during modeling activities

In the current state of the art, three distinct perspectives have emerged, aiming to address the challenge of data scarcity. As illustrated in figure 2.6, these perspectives focus on different levels of implementation and offer potential solutions to mitigate the problem, taking into account whether they train or utilize pretrained models or employ fine-tuning techniques. The details of the work done for each perspective are presented in the following sections.

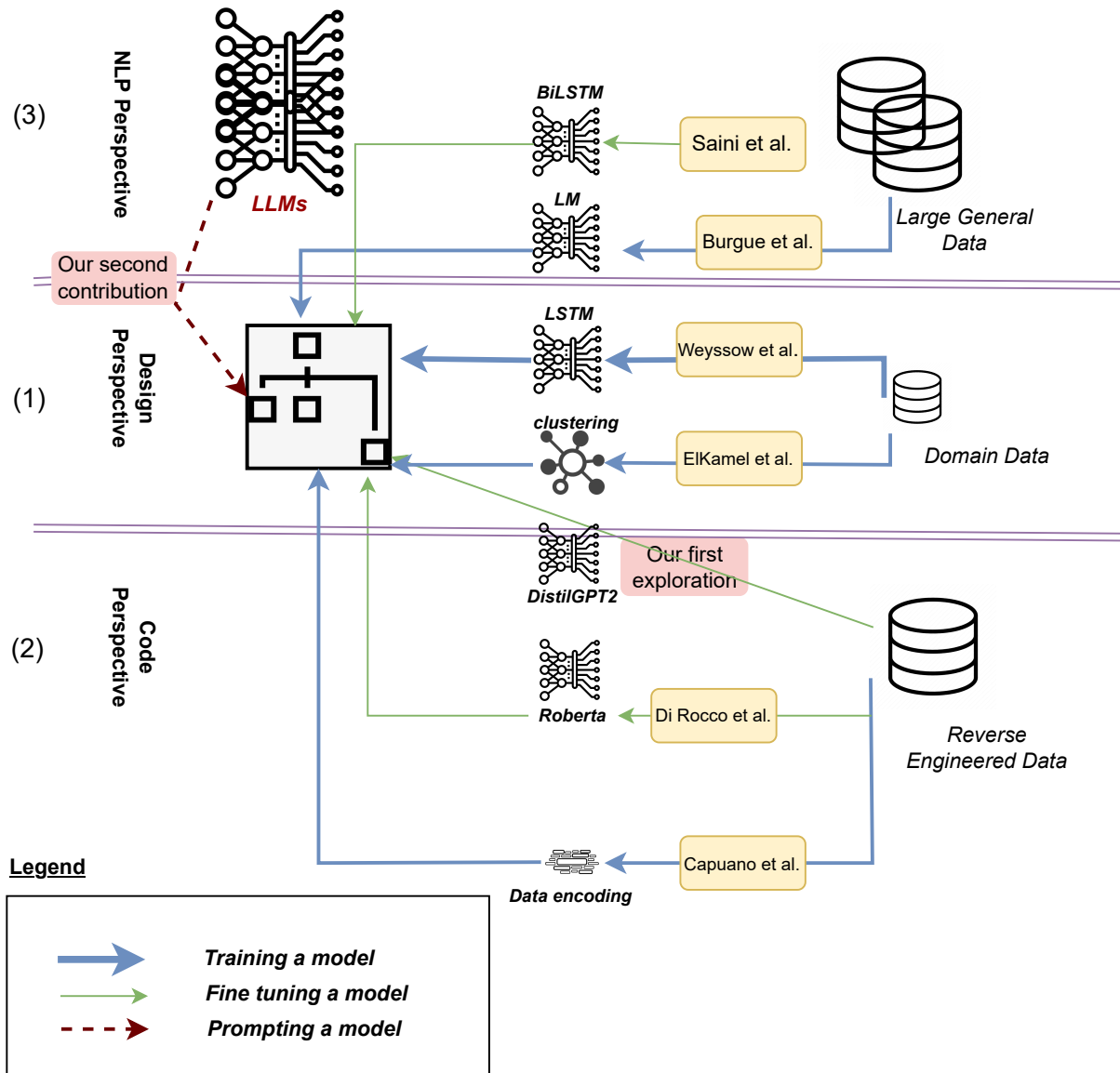


Fig. 2.6. Overview about the approaches in the literature

2.2.1. Leveraging domain models data for model completion (1)

The first perspective proposes embracing the limited amount of available models or metamodels and using them to train recommender systems. Weyssow et al. [50], gathered a dataset of models and metamodels relevant to the target objective and applied appropriate transformations to obtain textual formats and then train an LSTM neural network. The authors obtained limited results, especially when applied to the iterative construction of a metamodel. Other techniques of model completion does not involve sophisticated neural network models and use similarity criteria to suggest elements that are similar to the item being modified. This is the work of ElKamel et al. [22], which proposes related UML classes from a source of UML class diagrams. Using a clustering algorithm which classify a set

of UML classes into a large number of similar groups, the authors were able to index this dataset and use it to suggest related items.

2.2.2. Utilizing code data for model recommendation (2)

From a second perspective, Capuano et al. [16] trained a model recommender with data extracted from Java projects but they used document embedding techniques to abstract a set of domain concepts and suggested to the designer the most similar documents.

Similar to this work, Di Rocco et al. [37], exploited the available large code bases to reverse-engineering a set of models to train a RoBERTa language model. The results are acceptable, but the reverse-engineered models on which the authors had to rely for training led to suggestions that reflect implementation aspects rather than the modeled domains.

2.2.3. Extracting domain models from natural language sources (3)

The third perspective is to exploit what is already encoded in natural languages. Arora et al. [6] propose an approach to automatically extract domain models from natural-language requirements using dependency parsing and rule-based techniques. The study shows that the proposed approach is effective in extracting domain models from requirements documents and can be useful for specifying the system under development. According to expert evaluation, the extraction rules in their implementation achieve correctness between 74% and 100%. Saini et al. [39] proposed a bot to assist modelers with the creation of domain models from requirements expressed in natural language using a combination of NLP techniques. For learning the context information, the authors used a pretrained Bidirectional Long Short Term Memory (BiLSTM) neural network. Although their results show that the bot can be useful, they did not exploit LLM, which we believe will have a positive impact in predicting new modeling elements. And with the same goal of exploiting knowledge captured in general and specific natural-language documents, Burgueño et al. [15] exploited these documents to train language models to suggest model completions.

We believe that exploiting natural-language sources is a good idea to overcome the scarcity of the data to exploit deep-learning to assist in modeling activities. However, this work requires to train a language model from scratch for each specific domain, which remains a challenging problem in many scenarios. Besides, all the aforementioned approaches have been custom-designed from the ground up for specific modeling or specific modeling formalism, resulting in significant time and effort required for their development (e.g., limited reusability), and integration.

2.3. Conclusion

In this chapter, we have provided the foundational background information relevant to our thesis. We have covered key concepts and techniques that form the basis of our research. Additionally, we conducted a thorough literature review to explore the latest advancements and related works in the field. Building upon this foundation, the next chapter focuses on our initial exploration. We delve into the code perspective and investigate the usage of code source data for model completion.

Chapter 3

Exploring the Usage of Code Data for Model Completion

Introduction

This chapter presents the outcomes of our investigation into the usage of code data for model completion. We begin by clarifying the objective of this study and the approach we pursued. Subsequently, we detail the components involved and outline the different steps. Then, we present a preliminary evaluation of our approach to assess its correctness and validity. Finally, we conclude this chapter by summarizing our findings and reflecting on the efficiency of our approach.

3.1. Context and Proposed Approach

With the rise in popularity of object-oriented programs, it is more vital than ever to learn about data mining techniques to discover knowledge, patterns, and relationships between elements and as discussed in a previous section 2.2, one approach to address the modeling completion task is to use the code base as a starting point and extract the necessary knowledge to prepare a corpus for a recommender system.

In line with this, our research explores this path by following a series of steps that involve applying natural language (NL) components in the preprocessing and data cleaning phase. These steps are designed to extract meaningful information from the code base and utilize it to train a recommender system effectively.

Besides, our approach intends to be multi-objective by enabling the suggestions of two key elements:

- (1) *Suggesting related class names for elements or concepts that already exist in the partial model.* This task involves text generation, where the system generates suggestions for

class names based on the given context and existing model by leveraging advanced techniques.

- (2) *Predicting the association type between two introduced class names.* This task involves analyzing the relationships between class names and determining the appropriate association type that connects them (inherits from, composition and simple association). By employing machine learning algorithms, such as Random Forest, we can infer and predict the association type based on the encoded features of the class names.

Such tasks should help in constructing meaningful and accurate class diagrams, guiding the modelers in their decision-making process. It is important to note that both approaches follow a similar overall process, as illustrated in 3.1.

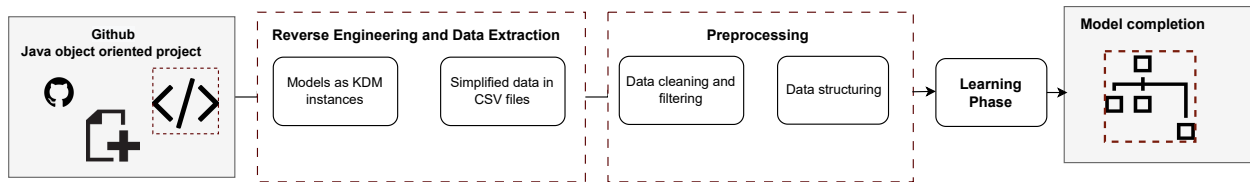


Fig. 3.1. Illustration of the process used to do model completion starting from Java code

The initial step of data gathering remains the same for both tasks. However, what differs are the subsequent steps in the preprocessing phase and the specific models, given the distinct objectives of each task. In the upcoming sections, we will delve into the details of each step in the pipeline.

3.1.1. Reverse Engineering and Data Extraction

We propose as a first attempt to implement an approach based on Model driven engineering principles to mine code repositories and extract knowledge in order to *provide the appropriate output to be fed to machine learning models for an efficient training or fine tuning.*

This approach serves as a crucial step in the process of model or design completion, as it allows us to uncover valuable information embedded within the codebase. Previous attempts were done in order to achieve Mining Software Repositories (MSR) [23]. In this work, we use the GitHub Java Corpus [5], which is a set of Java projects collected from GitHub. Before proceeding with the entire process, it is necessary to download each project locally using the provided GitHub URL. In Figure 3.2, we present a comprehensive depiction of the pipeline used later, illustrating all the process’s phases.

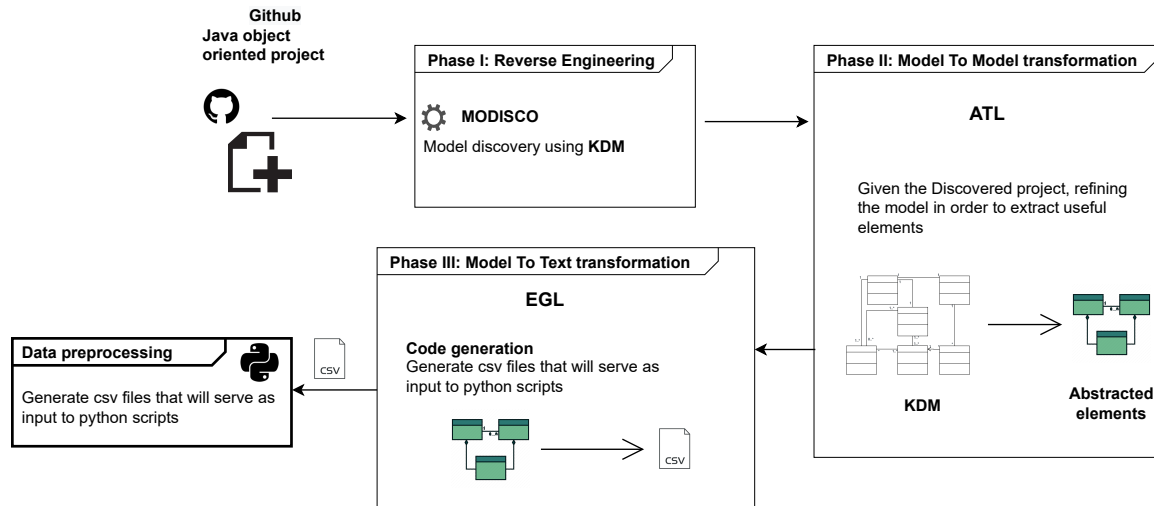


Fig. 3.2. Illustration of the process used to encode concepts of a Java repository.

First, to discover the representation of design elements within a given project, we rely on the powerful MoDisco tool [8, 14, 2]. MoDisco excels in reverse engineering legacy systems and stands out as an expandable open-source project. As an MDE framework, it possesses the capability to reverse engineer various types of models, including the highly versatile KDM representation [8]. One of MoDisco’s significant advantages lies in its extensibility, which empowers us to tailor the reverse engineering process to fit our project requirements.

We then go through an abstraction step of the discovered model where we keep and aggregate features depending on what a given scenario requires and for that we use a **model to model** transformation using Atlas Transformation Language (ATL) [43]. With ATL, we can define transformation rules that specify how the original model should be modified and transformed into an abstracted version. These rules facilitate the extraction, aggregation, and refinement of relevant features, ensuring that the resulting model aligns with the specific needs and objectives of the scenario at hand. In other words, we implement the right rules to extract from the initial reversed engineering models, instances of the KDM metamodel, elements like class names, edges or associations between classes and attribute names.

The final step is to generate the appropriate input that will be used to train the recommender system later. In this phase, we use an MDE tool (EGL) to automatically generate csv files from a processed model. EGL¹ is " a **model-to-text** transformation (M2T) language that can be used to transform models into various types of textual artefact". The format of the created files is based on the requirements of our recommender system.

¹<https://www.eclipse.org/epsilon/doc/egl/>

At the end of this pipeline, we meticulously extract data from a complex representation, converting it into CSV files. This carefully extracted data is subsequently employed in both previously explained tasks (3.1) with the purpose of assisting modelers in the process of creating class diagrams.

3.1.2. Preprocessing and Learning Phase

We recognize that the content of the obtained CSV files may not always be clean. A crucial preprocessing step is then mandatory to enhance the results of both tasks; classification and text generation.

3.1.2.1. Data Cleaning.

To make sure the data are good quality and appropriate, we carefully filter out irrelevant entries and remove any noise that might impact the model’s performance. For instance, in Java programming, packages play a crucial role in structuring code by assembling interrelated classes, interfaces, and supplementary resources. This grouping is orchestrated according to shared functionality or a unifying objective. The name assigned to a package holds substantial significance as it often offers insights into the nature and content of the encapsulated objects. As a result, one key principle for data cleaning is to consider only packages that are not related to Java code.

By excluding Java-related packages, we ensure that the generated concept suggestions align closely with the requirements of class diagram design.

Furthermore, we remove entries that contain code-related concepts or digits. Code-related concepts are typically more specific to the implementation details and may not be directly applicable to the high-level design considerations that class diagrams aim to capture. Similarly, removing entries with digits helps eliminate any potential noise that might arise from numerical values or identifiers.

3.1.2.2. Data Encoding.

Regarding the task of *generating related concepts or class names*, we prepare the remaining entries for appropriate input into the model, we structure them in a suitable format. Specifically, for each package, we organize the entries into pairs of class names, denoted as (className1, className2). This structure allows the text generation model to understand the relationship between different class names and generate concept suggestions that reflect the desired connections in the class diagram.

However for the task of *predicting associations type*, we follow a list of preprocessing steps as illustrated in fig3.3. Three processes are applied sequentially: (1) tokenization, which reduces the identifiers to single words; (2) stop words removal, which gets rid of common terms like the, to, by, etc. that don’t have much lexical significance.

(3) and lemmatization, which converts a word's several inflected forms into a single, "lemma," or root form.

The cleaned texts are then fed into the word2vec learning algorithm to leverage the distributed representations of words from a large corpora of text data and more specifically from the google news corpus.

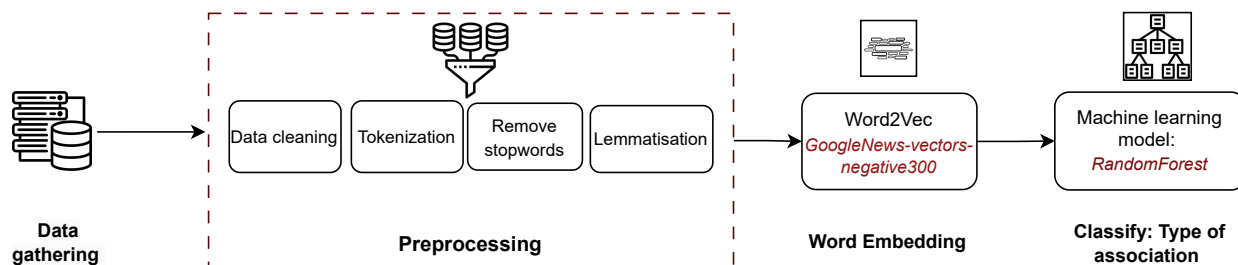


Fig. 3.3. Illustration of the process used to identify the association type between two concepts

3.1.2.3. Learning Phase.

We focus on training the text generation model to first generate concept suggestions. For this purpose, we employ the powerful GPT2² model, which has been pretrained on a vast English dictionary. By leveraging the pretraining on such a comprehensive linguistic resource, the model exhibits a strong understanding of language patterns and semantics. More precisely, the DistilGPT2 model is actually utilized, as a condensed variant of the GPT2 model [40]. It provides a solution that is more lightweight and efficient while retaining a high level of performance. This model has also been pretrained on a large corpus of textual data, which includes a variety of linguistic patterns and semantics.

To optimize the DistilGPT2 model for the task of generating concept suggestions, we **fine-tune** it on our prepared dataset. During the process of fine-tuning, hyperparameters such as learning rate, batch size, and number of training stages are considered. By adjusting these hyperparameters with care, we optimize the model's performance and guarantee efficient concepts generation.

When we address the task of predicting relationships between concepts, we employ the Word2Vec embedding technique, which is pretrained on extensive Google news web pages. We then employ as classifier the Random Forest ³ [12] relying on its ability to learn and capture the patterns and relationships present in the data.

The input to the classifier consists of the results obtained from encoding the concepts using Word2Vec embeddings. Random Forest is a versatile and robust machine learning algorithm that excels in handling high-dimensional data and capturing complex

²<https://openai.com/research/gpt-2-1-5b-release>

³<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

relationships. By combining the Word2Vec embeddings and the Random Forest classifier, we enable our system to effectively predict meaningful relationships between concepts. This assists in the accurate construction of class diagrams, where the model can provide insights and suggestions based on the discovered relationships between the encoded concepts. Combining word2vec and a certain classifier was especially used in previous research work for sentiment analysis, text categorization, and named entity recognition [29].

3.2. Evaluation and Discussion

In this section we present a preliminary evaluation of this intuitive approach. We start by explaining the setup of the experiments then we discuss the obtained results.

3.2.1. Setup

The evaluation setup involved the dataset extracted from the source code comprising a diverse range of class names and their associated contextual information. The dataset was divided into a training set (80%) and a test set (20%) to facilitate model training and evaluation. During the training phase, the DistilGPT-2 model was fine-tuned using the training set to capture the relationships between different concepts and generate contextually relevant suggestions. Notably, the subsequent key parameters were meticulously employed:

- **Learning Rate:** This parameter, pivotal in regulating the model’s adaptability, was set at 2×10^{-5} .
- **Number of Training Epochs:** The model underwent training iterations for a total of 5 epochs.
- **Weight Decay:** Employed as a preventive measure against overfitting, the weight decay was set at 0.01.

These parameters collectively shaped the training regimen, significantly influencing the model’s convergence speed, generalization capability, and susceptibility to overfitting.

Besides, to evaluate the coherence of the generated concept for class diagram design, a manual approach was adopted that aimed to simulate the work of a designer and replicate an incremental design process. Tasks were conducted in stages, mirroring the iterative nature of the design process. The initial stage involved providing the model with a partial class diagram or a set of existing class names. The model then generated concept suggestions based on the given context. Then these suggestions were utilized to augment the class diagram incrementally, reflecting the natural progression of design activities. Finally, human evaluators, who are familiar with class diagram design, investigated the generated concepts and assessed their coherence and contribution to the obtained diagram.

Similarly, the evaluation of the association type classification task was conducted using a carefully designed setup. The dataset was again divided into training and test sets, with 80% of the data allocated for training purposes and the remaining 20% reserved for testing the model’s performance.

To ensure robustness and mitigate biases, the dataset was stratified during the splitting process to maintain a balanced representation of different association types (inherits from, composition and simple association) in both the training and test sets. And as explained previously, the classification task was approached using the RandomForestClassifier with 200 estimators. Each estimator in the ensemble is essentially a decision tree that independently predicts the association type, and the final classification result is based on the collective prediction of these trees.

This approach helps in capturing the generalization capability of the model across various association types. A replication package with the code of experiments can be found on our Github repository [17].

3.2.2. Results

3.2.2.1. Suggesting concepts:

The evaluation of the method centered on determining the accuracy and coherence of the generated concept suggestions for class diagram design. In addition to the low training loss of 0.64, the evaluation of the approach also revealed a test loss of 0.55, which is in close proximity to a random baseline. This indicates that the model’s performance on unseen data is not significantly better than random guesswork.

As part of the evaluation process, an attempt was made to simulate the work of a designer by conducting tasks in stages and evaluating the generated concepts using the trained model. However, the simulation’s results were far from satisfactory. The generated suggestions lacked coherence, failed to offer meaningful insights, and did not contribute to the design process.

3.2.2.2. Classifying association types:

We assess the classification performance of the association type prediction task. Upon evaluation, the model exhibits a training accuracy of 0.84 and a test accuracy of 0.78. More precisely, our model demonstrated precision rates of 0.82 for inheritance, 0.76 for simple association, and 0.66 for composition. The recall values were 0.64, 0.89, and 0.75 respectively, showcasing the model’s ability to accurately identify each association type. However, it is important to note that the obtained results are not as satisfactory as desired. While the model demonstrates a moderate level of accuracy, further improvements are necessary to enhance its predictive capabilities.

Association type	precision	recall
inheritance	0.82	0.64
simple association	0.76	0.89
composition	0.66	0.75
accuracy		0.78

Tableau 3.1. Association type prediction evaluation

3.3. Discussion

We observed two primary categories of errors:

- **Coherence Issues:** The generated concept suggestions sometimes lacked coherence and failed to provide meaningful insights for class diagram design. These suggestions appeared disconnected from the context, impacting the quality of the generated diagrams.
- **Insufficient Contribution:** The suggestions often fell short in contributing effectively to the design process. They failed to offer valuable suggestions that a designer could readily incorporate into the diagram, impeding the model’s ability to enhance the design workflow.

One of the reasons for these poor results is the inherent nature of the domain, which is a vast and diverse open space with numerous design possibilities. It becomes difficult to restrict design options and obtain results that are similar to the initial model that serves as the ground truth. During the preprocessing phase, the presence of a large quantity of messy data related to code, such as *utilities*, *repositories*, *controllers*, and other code-related artifacts, posed a significant challenge. Despite efforts to remove such noise, we discovered that there were too many of them, making the removal impractical.

As a matter of fact, the efficacy of the machine learning model is heavily dependent on the quality and representativeness of the training data. If the training data fails to adequately represent the wide variety of concepts and relationships pertinent to class diagrams, the generated suggestions may be limited or biased. In addition, the model may have difficulty with concepts that are uncommon or poorly represented in the training data. Furthermore, despite being a distilled variant of GPT2, the DistilGPT2 model still requires substantial computational resources for training and inference. The process of fine-tuning the model can be time-consuming, and real-time generation may incur additional computational overhead.

3.4. Conclusion

In light of the limitations and difficulties associated with relying solely on code data for model completion in class diagram design, we decided to investigate an alternative strategy. Instead of significantly relying on code data, we will directly leverage LLMs for concept

generation and completion. By utilizing LLMs, we can access a broader range of textual resources and knowledge bases, enabling us to generate concept suggestions that are more exhaustive and contextually accurate. Particularly, we will investigate the application of few-shot learning techniques, which enable the model to learn from a small quantity of training data, thereby making it more adaptable to various formalisms and scenarios.

Chapter 4

Few-Shot Prompt Learning for Automating Model Completion

Introduction

In this chapter, we present an innovative method for automating model completion through the utilization of powerful left-to-right Large Language Models (LLMs). Firstly, we provide an overview of the context surrounding our research contribution. Secondly, we introduce our novel approach, which represents a significant departure from traditional methods. We then delve into the specific details of how our approach was applied within different modeling formalisms. By delving into these aspects, we aim to provide a comprehensive understanding of our innovative paradigm in automating model completion. We conclude with a preliminary evaluation to highlight the potential and effectiveness of our approach in assisting designers in model completion tasks.

4.1. Context and Motivations

Given the scarcity of domain-specific data in modeling, one other promising approach to assist modelers in their task is to exploit the large amount of encrypted learned general knowledge contained within the LLMs instead of relying on domain-specific training data and training a new model or fine tuning a pretrained one (section 2.2, figure 2.6).

In order to accomplish this, we leverage semantic mapping, a paradigm for extracting knowledge from text and aligning it with elements in modeling formalisms. This method serves as a crucial bridge, allowing the system to not only comprehend but also eventually build meaningful representations within the modeling formalism. This dynamic process facilitates the transformation of linguistic complexities and contextual insights from natural language into a structured framework, thereby seamlessly integrating textual comprehension with formal representation.

4.2. Approach

As mentioned in previous sections, the main goal of our study is to complete a model under-construction (a.k.a. partial model) by suggesting related elements. In this approach, we specifically take advantage of GPT-3, which is one of the most powerful LLMs—it contains 175 billion parameters [13]—, to represent most of the existing general concepts to support software specialists when modeling. To adapt the semantics of the general concepts represented in GPT-3 to the semantics of modeling formalisms, we use as mentioned before the **semantic mapping**, an information extraction task where the goal is to produce a structured meaning representation from a natural language input.

As a matter of fact, we suggest to use two semantic mappings as shown in figure 4.1: one that takes the model under construction (i.e., the input to our system) and builds the prompt, and another one that obtains model completion suggestions from the text produced by the language model. These semantic mappings rely heavily on the targeted modeling formalisms.

We apply a first semantic mapping, i.e, we construct a text representation that will serve as input to GPT-3. We then query the chosen Large language model which returns a textual output that follows a certain pattern. We finish by applying another semantic mapping—in particular, a parsing—to the obtained text and extracting relevant model elements by applying suitable text transformations. It is important to highlight that the process described is applied in an *iterative* manner to allow for continuous refinement and completion of the model.

We illustrate our approach with two examples coming from two categories of modeling languages: static models, i.e., UML class diagrams; and dynamic models, i.e., UML activity diagrams. By considering multiple formalisms, we aim to gain a more nuanced understanding of our method’s performance, applicability, and generalizability.

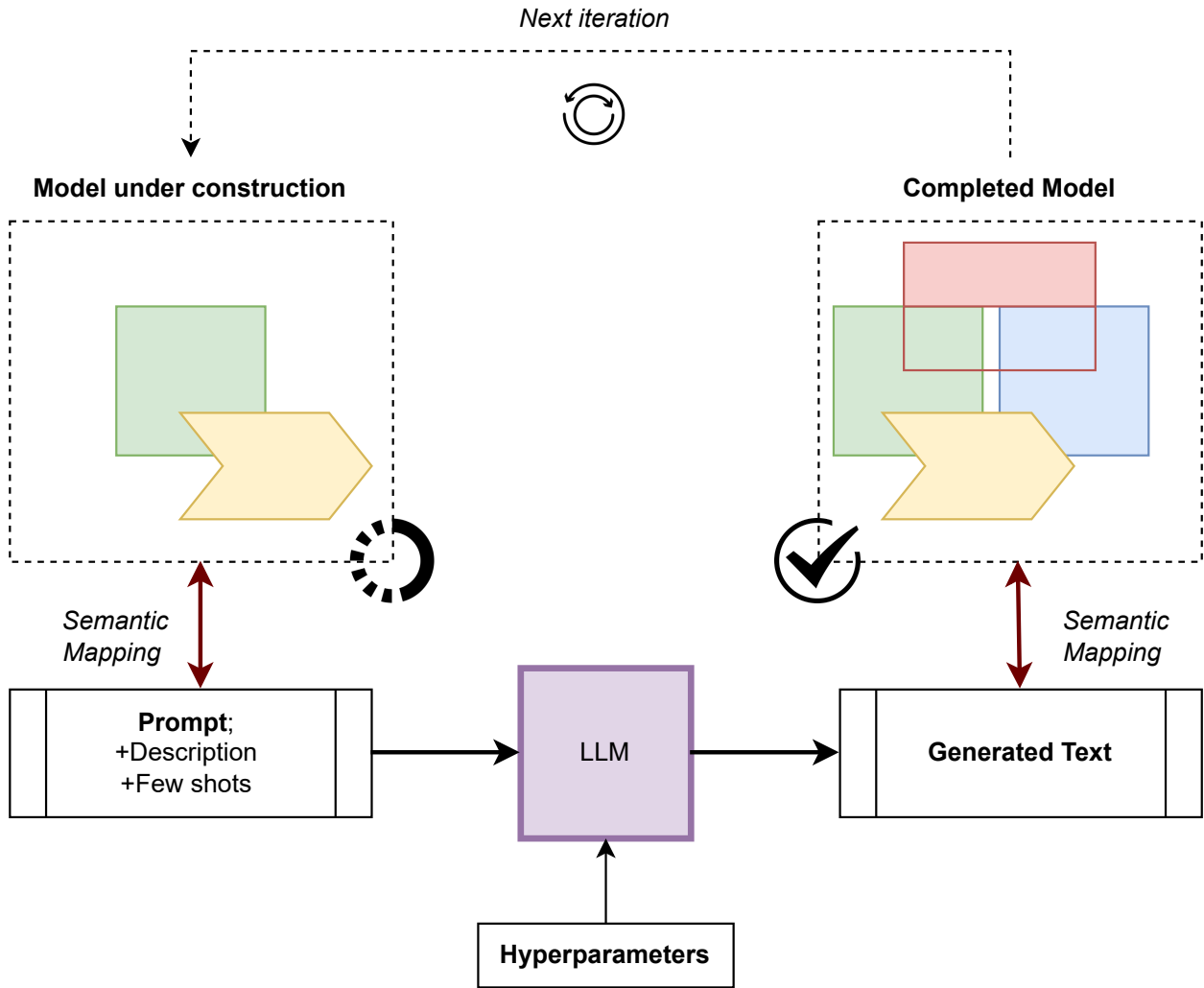


Fig. 4.1. Overview of the approach: using few shot learning to automate modeling completion -one iteration-

4.2.1. Static diagram completion: Class diagrams

Using static diagrams for domain modeling usually consists of representing the domain entities, their properties or features and their relationships.

The UML class diagrams shown in Figure 4.2 is a partial description of a banking system. It represents the partial domain model that is already defined by the user.

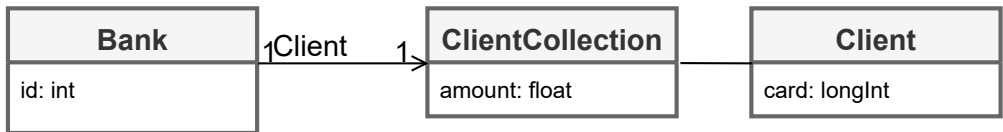


Fig. 4.2. Example of domain model: Bank class diagram - Partial model

4.2.1.1. Class Names suggestions.

We focus first on how we design our completion system to suggest entities, i.e., new classes. The core of our approach centers on creating a thoughtfully designed prompt which plays a key role in guiding our model’s completion process. We start by incorporating the task description which involves suggesting concepts or class names that are connected or related.

Then, we provide the LLM with a set of shots or examples that act as guides for the model’s reasoning process. They enable the model to generalize from the presented instances and make informed predictions about the relationships, connections, and concepts that are likely to exist within the partial domain model.

More specifically, we craft the prompt upon existing diagrams from diverse, unrelated domains. We introduce first the package name (X) to establish the contextual backdrop of this domain mode, then we put pairs of related classes (Y and Z where Y and Z are connected with an association) and, to follow a certain pattern where we introduce the relationship between two related elements, we represent them between square brackets.

We obtain this format X: [Y,Z], [Z,C]... Figure 4.3 shows the few-shots that we provide for our example. In the final step, we build a query from the partial domain model under construction. To do this, we select between 2 and 4 pairs of related classes, put the classes names in square brackets and add them to the prompt. In Figure 4.3, under “Generated text” we can observe in bold the text that has been generated for our example.

Prompt:
Generate related concepts:
Hospital: [Nurse, Staff], [Department, Room], [Nurse,patient], [Nurse,department]
ReservationSystem: [SpecificFlight, GeneralFlight], [Airport, City], [passenger, plane], [trip, passenger], ...
... (*Three more shots*)
Bank: [bank, client], [client, clientcollection]

Generated text:
Bank: [bank, client], [client, clientcollection], [**loan, clientcollection**], [**account, Savings**], [**account, Checking**]

Fig. 4.3. Prompt and generated text for class names and association prediction to complete the Class Diagram of Fig. 4.6

We apply a string-searching algorithm on the generated text to extract relevant class names and the association that exist between them; we also remove spelling errors and noisy data such as digits, which are usually not part of domain models. In our example, after this step, we obtain that potential missing classes (and associations between them) are **Checking**,

Savings, **Account** and **Loan**. These classes are suggested to the user as shown in Figure 4.6 with a red color. In section 4.2.1.3 we explain how we consider the suggested relationships between the new concepts and introduce them in the model under construction. One can notice that for this example all the suggestions are closely related to the banking domain that is being modeled.

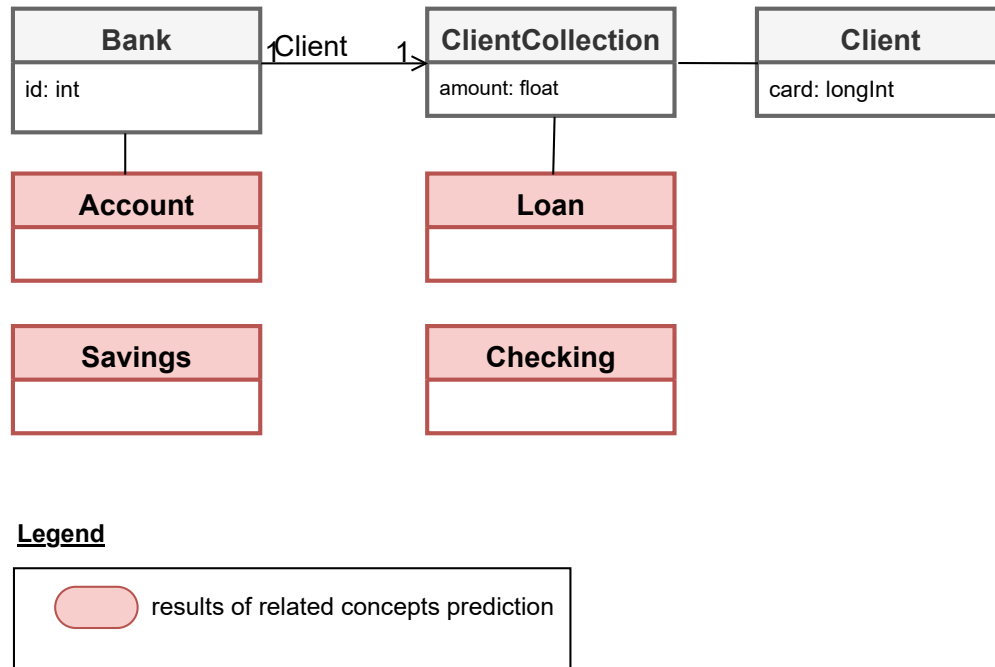


Fig. 4.4. Example of domain model: Bank class diagram - updated model with suggested concepts

In practise, our engine follows a ranking strategy to suggest new elements. We query GPT-3 several times with different prompts, where all the prompts have the same shots but different queries, each query containing a different subset of model elements from the partial model. As a result, we obtain for each prompt a set of suggested concepts. Then, all the obtained concepts from the different prompts are ranked by its frequency from higher to lower. Only those concepts with higher frequency are considered.

4.2.1.2. Attributes suggestion.

Given a partial model, to generate prompts for attribute completion, we concatenate the package name and existing class names with their attributes in square brackets, ending with the class for which we are finding potential attributes. Like we do for classes, we use a frequency based ranking function that takes as input all the text generated by GPT-3 for different prompts. Then, we generate attribute suggestions using those concepts which are at the top of the ranking.

Figure 4.5 illustrates the prompt and resulting text for the different classes in the model under construction in Figure 4.6. For instance, for the class Client, we have obtained `name`, `address` and `id` as potential missing relevant attributes.

Prompt:

Generate missing attributes for each class in this class diagram:

```
package company: employee: [id, name, lastName, occupation]; manager: [id, name, department];  
company: [name, holding] => employee: [id, name, lastName, occupation, department, experience,  
revenue]; manager: [id, name, department, team, revenue]; company: [name, holding, address, web-  
site]
```

```
package bank: bank: [id]; clientCollection: [amount]; client: [card]; Account[]; Loan[]; Checking: [];  
Savings: [] =>
```

Generated Text:

```
package bank: bank: [id, name]; clientCollection: [amount]; client: [card, name, address, id];  
Checking: [amount]; Savings: [amount]
```

Fig. 4.5. Prompt and generated text for attribute completion.

The process of completing the typing of each attribute involves providing the engine or LM with a set of attribute names along with their associated types. Then, to determine the relevant type for a specific attribute, we prompt the LM by providing the attribute name for which we want to know the type. By leveraging this information, the LM generates a response that recommends the most suitable type for the specified attribute.

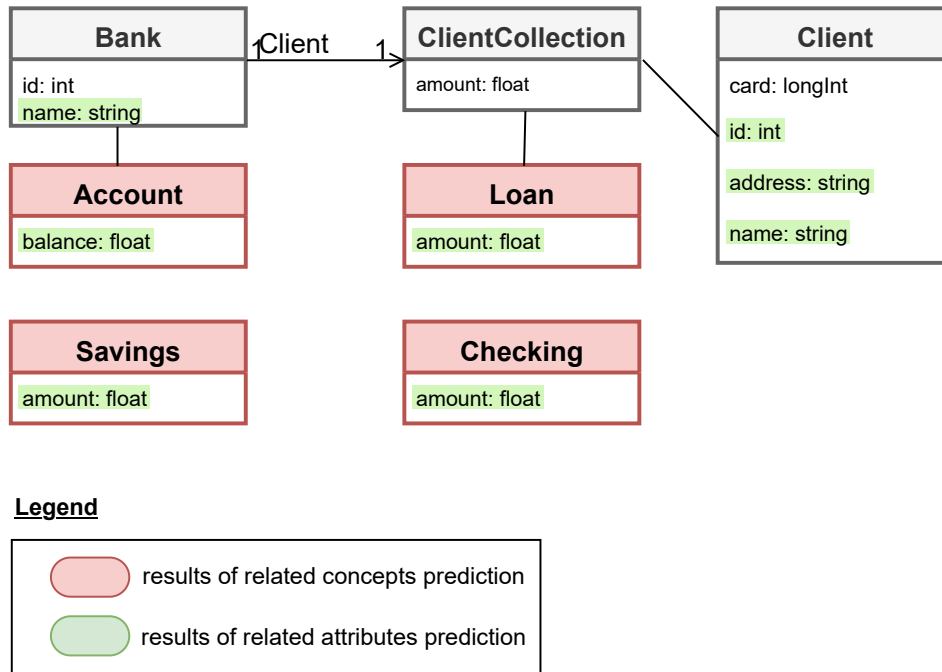


Fig. 4.6. Example of domain model: Bank class diagram - updated model with suggested attributes

4.2.1.3. Associations names and types suggestion.

Regarding the suggestion of association names, we design our prompt as follows; from unrelated diagrams, we select pairs of classes that have an association between them. Each pair of classes is used to build a shot by concatenating the name of the classes and the name of the association.

Prompt:
 Predict association name:
 employee, company => worksIn
 person, Home => owns
 car, driver => drives
 ClientCollection, Loan => **Generated Text:**
Owns

Fig. 4.7. Prompt and generated text for association name completion.

Similarly, when it comes to the suggestion of association types, our prompt is designed as follows: from unrelated diagrams, we select pairings of classes with an association. Each pair of classes is used to construct a shot by concatenating the class names and the association type.

Prompt:

Specify the nature of the association between these concepts, inheritance or association or composition or no:

student, person => inheritance

computer, cpu => composition

plane, passenger => no

person, address => association

Account, Savings =>

Generated Text:

inheritance

Fig. 4.8. Prompt and generated text for association type completion.

Figure 4.9 represents the completed model after going through the semantic mapping step to extract the knowledge (type and name of associations) and introduce them to the model under construction.

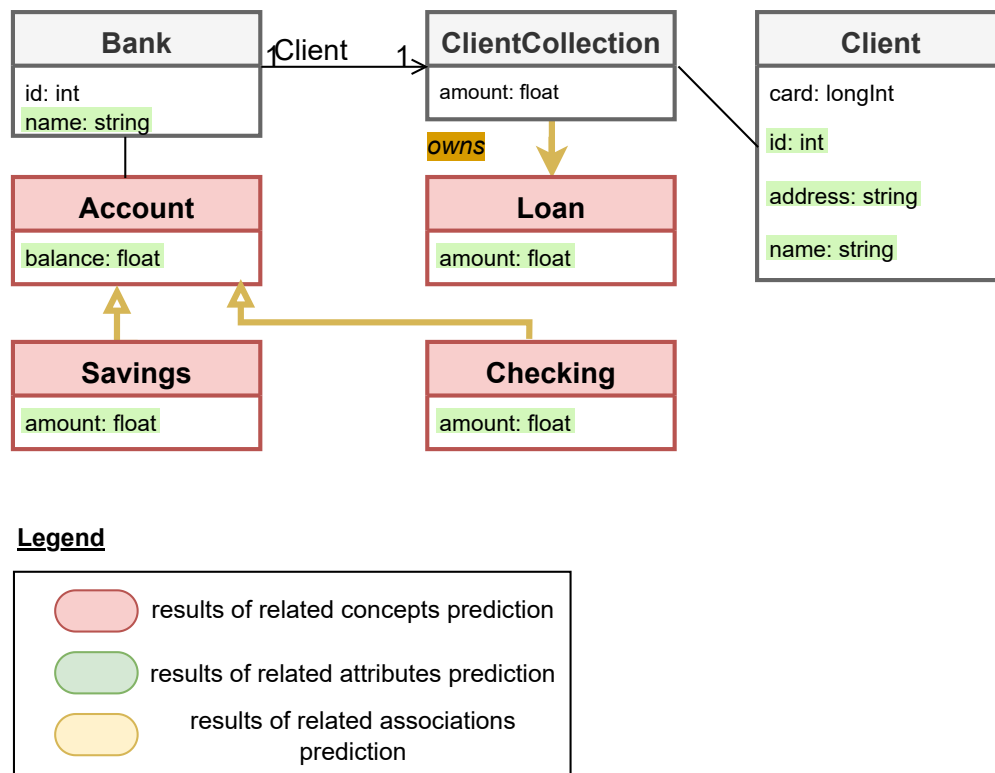


Fig. 4.9. Example of domain model: Bank class diagram - updated model with suggested type and names of associations


Mapping	
●	start
⦿	end
→	=>
◇	<i>condition1</i> / <i>condition2</i> => if <i>condition1</i> {...} else {...}
	action (as text)

Tableau 4.1. Semantic Mapping for Activity Diagrams

In this section, we demonstrated how it is possible to recommend various elements within the same modeling formalism (i.e. UML class diagram) following this paradigm. In the following section, we examine its effectiveness using a distinct modeling formalism.

4.2.2. Dynamic diagram completion: Activity diagrams

Since structural diagrams do not define sequences, any fragment can be used to generate the prompts to complete a diagram. In the case of dynamic/behavioral diagrams, such as activity diagrams [38], there are strong precedence/sequence constraints to consider (e.g., to represent time), as it can be seen in Figure 4.10. Hence, to apply prompt learning, we need to define shots and prompts in a way that they preserve those constraints.

In this section, we present how prompt learning can be applied for the completion of activity diagrams. Once again, we need to map the semantics of activity diagrams to a pattern that a LLM such as GPT-3 is able to understand and for which it provides meaningful results. To deal with precedence constraints, we designed our prompts and parameterised them to predict the next actions in a partial sequence¹. To build the prompts, we defined simple transformation rules to match the elements of the activity diagram to the appropriate keywords that conform the prompt that we send to GPT-3 as Table 4.1 shows. Note that so far, we have only focused on a subset of the activity diagram language.

To illustrate our idea, we introduce the example of an online shopping workflow. Figure 4.10 shows, in the upper part, the partial activity diagram that is already defined by the user.

To create our prompt, we design 3 shots using real activity diagrams extracted from a public repository [3], which have been mapped using the rules described above. Figure 4.11 represents the prompt for this example and the GPT-3 generated text.

After mapping the generated text into model elements, the elements that Figure 4.10 shows in white represent the completion elements that we obtain. The resulting completions are considered good from two different points of view. From a conceptual point of view, it

¹This implies that we need to set the GPT-3 `maximum_number_of_tokens` hyperparameter to a relatively low number—in our experiments it has been set to 50.

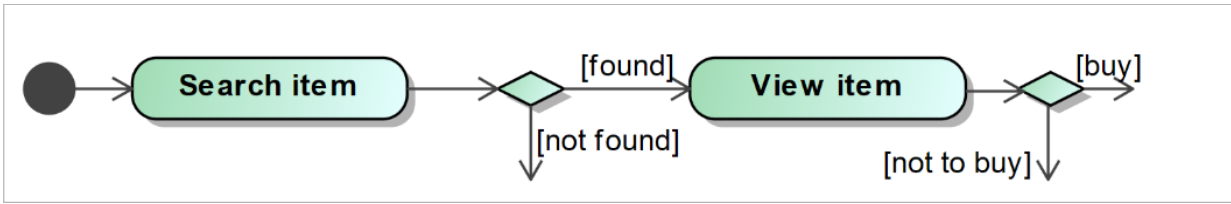
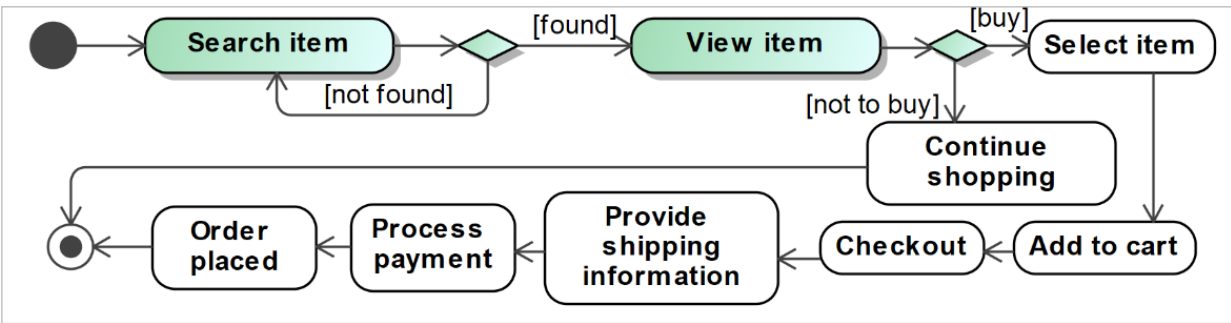


Diagram under construction (D)



Completed diagram (D')

Fig. 4.10. Example of activity diagram: Online Shopping

Prompt:
 Complete the workflow:
 Ticket Vending Machine: **start** => request trip info => provide trip info => process trip info => payment request => provide payment Info => process payment => pay with card / pay with cash => **if** with card {authorize card payment} => **end**.
 ... (two more shots)
 Online Shopping: **start** => search item => found / not found => **if** found {view item => buy / not to buy}}=>

Generated Text:
if buy {select item => add to cart => checkout=> provide shipping information => process payment => order placed => **end**} **else** {continue shopping => **end**}} **else** {search}.

Fig. 4.11. Prompt and generated text for the Activity Diagram of Fig. 4.10

fits perfectly the domain being modeled; and from a syntactic point of view, the completion suggested comply with the activity diagram syntax.

4.3. Preliminary evaluation

This is the initial step of our evaluation agenda. We focus in this part of this chapter on the correctness of this approach to do model completion and we present in the next chapter the evaluation part of the usefulness of using few shots learning to accomplish this task. So far, we have evaluated our idea on domain models represented as class diagrams from

a public repository. In the future, we are planning to evaluate this approach with other modeling formalisms such as activity diagrams and petriNets.

4.3.1. Setup

We followed a manual data sampling methodology, which consisted of selecting 30 domain models from a larger dataset called ModelSet [24]. The goal was to have a dataset with samples of different sizes and containing domain models covering multi-disciplinary domains such as education, finance, entertainment, etc. This manual work was required given the fact that the dataset contained numerous models representing source code and therefore containing low-level implementation details.

The format of the selected class diagrams is .ecore, thus we apply model transformation techniques to extract relevant details and search for valuable information. The MDE paradigm goes one step further in the lane of this step and advocates the use of metamodels and model transformations to speed up and simplify the task. We follow different strategies to simulate these scenarios and we compare the obtained results using metrics among the following ones:

$$\text{Accuracy} = \frac{\text{Correct named samples}}{\text{Total of all samples}} \quad (4.3.1)$$

$$\text{Recall} = \frac{\text{Number of matched elements}}{\text{Total of all relevant elements}} \quad (4.3.2)$$

$$\text{Precision} = \frac{\text{Number of matched elements}}{\text{Total of all recommended elements}} \quad (4.3.3)$$

We define "matched elements" as instances where a direct match or semantic equivalence exists between two elements. This includes scenarios where the elements correspond exactly or convey the same meaning, enhancing the inclusiveness of the matching criteria.

Based on OpenAI's assertion that the latest GPT-3 model, by the time we are conducting this work, text-davinci-002, has been trained on a larger dataset, we have chosen to employ it. This increase in training data contributes to the model's enhanced ability to generate more accurate results. Our choice is based on the belief that this enhanced training positively impacts our current work. While our approach utilizes a tuning-free prompting—which means the engine generates answers directly without adjusting the parameters of the pre-trained language model taking into account only the provided few shots—we still have to set certain hyper-parameters. One such hyper-parameter is the `temperature`. In general, GPT-3 tends to select words with a higher likelihood of occurring when the temperature is lower. To avoid limiting the search space, we set it between 0.70 and 0.90, as this range provides better results for more creative completions. Another important parameter is the `maximum_number_of_tokens` that can be generated by the model. In our approach, we set this parameter differently depending on the task. For class names and attribute predictions,

we set it to 20 as the goal is to generate several words and complete missing concepts, whereas for association name predictions, we set it to 1 as the goal is to generate one precise word.

Furthermore, we have automated the completion process by automatically simulating the behaviour of a modeler using our proof of concept tool. A replication package with the code of our tool and experiments can be found on our Github repository [18].

4.3.1.1. Class names suggestions. To evaluate whether our approach leads to an effective suggestion of new classes, from each model M_i , we took 20% of its elements as the already defined partial diagram M'_i and we simulated an incremental design process starting from M'_i . First, we performed a first round (R1) of completion suggestions, then we validated the suggested results manually adding to the model under-construction M_i those which were semantically equivalent to those elements in M_i . The accepted elements were included in the partial model M'_i , resulting in the partial model M''_i . Then, we performed a similar second round (R2) starting from the partial model M''_i . As mentioned before, in each round, we generated three prompts with which we queried GPT-3, each incorporating a varying subset of concepts from the incomplete model. Then, we employed a frequency-based ranking algorithm to determine the final suggested class name.

4.3.1.2. Attributes suggestion. To assess the effectiveness of our approach when suggesting new attributes within a class, we have selected randomly 212 classes from our dataset, have removed 75% of their attributes and have generated attribute completions for them. Note that this means that for classes with three or less attributes, we removed all of them, which was the case for most of the classes. To do so, we only perform one step because we notice that the number of attributes in most of the classes in our validation dataset is not high (average of 2). Once we obtained the completion suggestions from our engine, we manually approved those which are either exact matches or semantically equivalent elements to those in the ground-truth model.

4.3.1.3. Association names suggestion. We finally evaluate whether our approach is able to suggest meaningful association names. We extract from our dataset 40 pairs of concepts, where each pair contains the names of two associated classes. Then, we query 3 times our engine to suggest a name for each association, each attempt with the same prompt but a different temperature. Then, we use a frequency-based ranking function to suggest the final association name. We validated manually the output of our engine and approved those which are either exact matches or semantically equivalent to those in the ground-truth model.

4.3.2. Results

4.3.2.1. Class names suggestion. As explained previously, we collect results for two successive steps. Table 4.12 summarizes the precision and recall metrics for both steps. We

observed that the *Recall* improved from R1 to R2 while the *Precision* decreased slightly. This is due to the fact that the number of correctly suggested elements increases from one round to the next, while the number of incorrect elements increases, too.

	Precision R1	Precision R2	Recall R1	Recall R2
avg	0.57	0.56	0.29	0.45
std	0.26	0.24	0.18	0.25

Tableau 4.2. Results evaluating class names prediction

We also observed that domain models that resulted in the best results (recall between 0.8 to 1) were addressing very common domain/topics used by humans in natural language such as *banking*, *university* and *library*. Yet, models whose domains contained information that falls further from natural language—such as a model whose package name was `AUni`—resulted into poor results (a recall between 0 and 0.1).

4.3.2.2. Attributes suggestion. For attribute suggestions, we evaluate the recall, defined as the ratio of the ground-truth attributes being found in the Top-N recommended items. In the selected domain models, most classes contain a very limited number of attributes, thus we are only considering the recall metric to check whether we are able to obtain these missing attributes.

The average recall is 0.7 with a standard deviation of 0.4, which can be considered a promising result.

4.3.2.3. Association names suggestion. An interesting metric to evaluate these suggestions is the accuracy, defined as the ratio of correctly predicted association names with respect to the total suggestions. This is, unlike before, we are no longer interested in recognizing the relevant elements, but checking how many times the engine was correct. We have obtained an a precision of 0.64, which also seems promising.

Although the experiments were conducted on distinct datasets and various metrics were employed to evaluate both approaches for completing and predicting missing elements of class diagrams, a comparative analysis is drawn against the outcomes presented in the work done by Weyssow et al. [50] and our suggested approach as illustrated in 4.14. This comparison unmistakably highlights the superior accuracy of our approach, notably excelling in the prediction of attributes and associations.

	Recall	Precision
Classes	0.45	0.57
Attributes	0.7	-
Associations	-	0.64

Fig. 4.12. Results evaluating our approach

	Recall@20
Classes	0.52
Attributes	0.51
Associations	0.47

Fig. 4.13. Results in terms of Recall@20 in the work of Weysow et al. [50]

Fig. 4.14. A comparative display of findings in two Independent works

The direct comparison highlights the advancements achieved through our proposed methodology, validating its efficiency and accuracy in completing and predicting class diagram elements.

4.4. Conclusion

Existing work focuses only on static model completion and requires a large number of training examples to be effective. With this novel idea, we show that we can improve diagram completion without the need of training or fine-tuning LMs with large sets of examples. We additionally show that few-shot prompt learning can also be applied to the completion of other domain modeling diagrams such as behavioural ones that are more constrained semantically. In the next chapter, we focus on evaluating the usefulness and effectiveness of this proposed approach through conducting a user study.

Chapter 5

On the usefulness of Prompt Based Modeling Assistance

Introduction

In this chapter, we begin by introducing a graphical editor that we have developed as a tool to conduct a user study. This graphical editor serves as a platform for implementing the prompt-based modeling assistance methodology that we aim to evaluate. Next, we present the results of the user study conducted to assess the utility of the proposed approach. We analyze and discuss the feedback received from the participants, highlighting their perceptions of the usefulness of the modeling assistance system. Additionally, we explore any challenges or limitations identified by the participants, which can guide future improvements and refinements of the recommendation system. We finish the chapter with a brief conclusion.

5.1. Tool support

To implement our approach we thought about generating a tool for our own Domain Specific Language, i.e a Graphical Editor for completing class diagrams. Creating a DSL in the MDE context is often called Domain-Specific Modeling (DSM). By employing the Eclipse platform, specifically the Sirius plugin [4], we leverage the powerful capabilities of these tools to construct our DSL-based graphical editor. In fact, Sirius offers a solution for the quick development of graphical tools without any knowledge of the backend procedures and is very flexible and adaptable[49]. Obeo, the firm that developed Sirius, offered a list of best practices[1] that we adhered to, in order to assure better performance.

5.1.1. Developing a DSM Graphical Editor for class diagrams

To create this tool, we follow several key steps. First, we generate a metamodel using EMF Ecore, which defines the structure and behavior of the editor. The metamodel serves as a formal representation of the tool's concepts and their relationships, enabling visualization and manipulation of class diagrams. Figure 5.1 illustrates the structural meta-model, showcasing the incorporated features and elements.

Next, we define notations for the editor by creating a Sirius representation model that serves as a mapping between the metamodel elements and the graphical representations, such as shapes, colors and connections, establishing then the visual appearance of the editor. This mapping is typically defined through the ODesign file, short for "Sirius Object Designer" File. It contains configurations, rules, and instructions that link the elements of the metamodel to their corresponding graphical representations. For instance, in our ODesign file we ensure that class elements, depicted in the metamodel by the UML class "Clazz" (5.1), are represented as rectangles with grey colors and we orchestrate the placement of the attributes elements within these rectangular class elements.

We also specify the behavior of the editor, encompassing various editing capabilities like adding, deleting and modifying elements. Our careful configurations of the ODesign file contribute to a unified and intuitive user experience.

Finally, we generate the necessary code and configurations to bring the editor to life, ensuring that it is fully functional according to some specified specifications. Throughout this process, we incorporate custom code to modify the editor's behavior to align with specific requirements and constraints, thus enhancing its functionality and adaptability.

5.1.2. Recommendation modes

The next step is to create the recommendation module. We apply the approach described in chapter 4. In order to integrate it seamlessly with the existing Java-based editor, we utilize Java code for making the API calls to OpenAI¹. Although OpenAI's API does not provide specific Java code for these calls, we leverage a different plugin² that allows us to avoid the need for integrating Python scripts and ensures both compatibility and efficiency.

The editor offers 3 recommendation modes;

- (1) In the **Recommendation On Request Mode**, the user has the flexibility to request specific types of predictions during the modeling process. This mode allows him to actively seek potential attributes, concepts, or associations based on his needs. As shown in figure5.2, with a right click on a specific class in the canvas, a list of

¹<https://platform.openai.com/playground>

²<https://github.com/TheoKanning/openai-java>

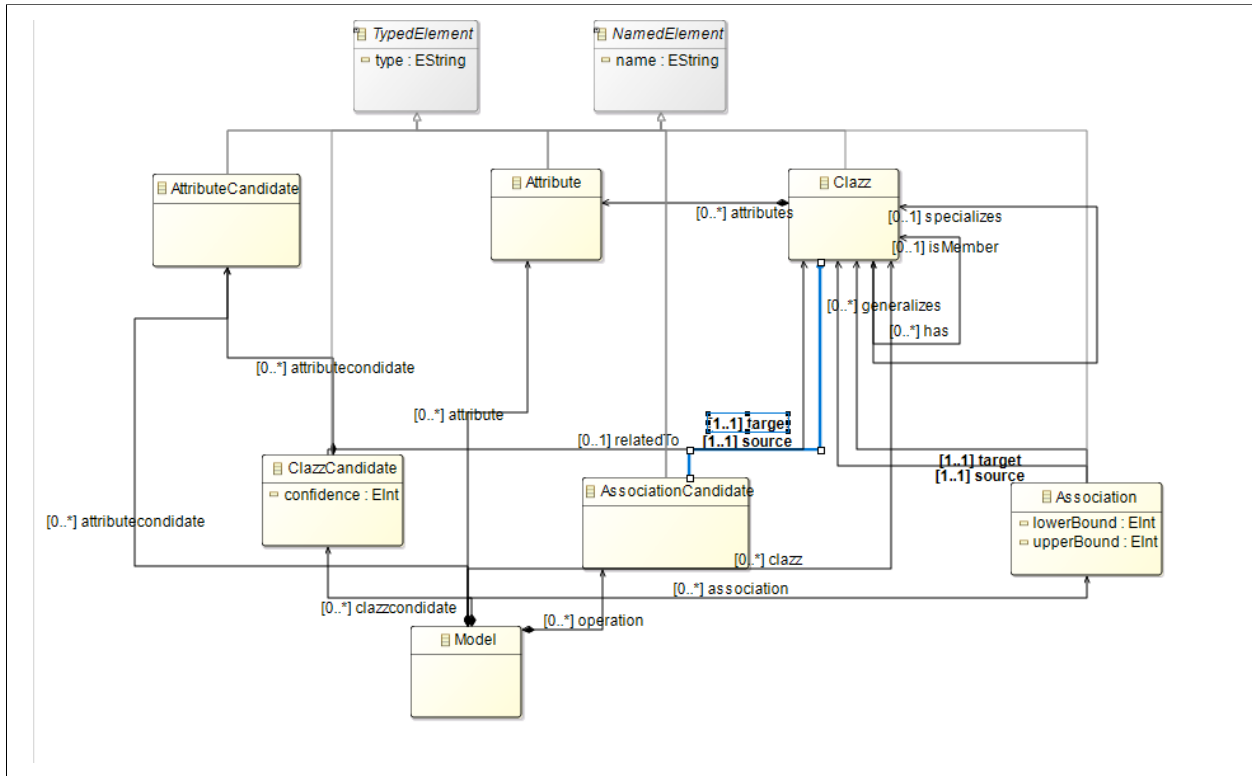


Fig. 5.1. Ecore metamodel of Class diagram editor

attributes to that class will be shown to choose from and be added to the model under construction.

- (2) In the **Automatic Mode** suggestions for concepts and operations are continuously provided as the modeling process unfolds. Whenever elements are added to the model under construction, an implemented algorithm triggers the suggestion of relevant elements. The canvas displays two lists, each containing suggestions for a specific type of element, as illustrated in the accompanying figure 5.3. These suggestions aim to assist modelers by offering timely recommendations that align with the ongoing modeling context.
- (3) The **Recommendation at the End Mode** is a mode that allows users to evaluate their completed modeling task and receive suggestions on any missing elements as a final step. In this mode, the user simply clicks on a button labeled "predict" when he believes he has finished the modeling process. The system then generates a list of suggestions that includes concepts, attributes, and associations for each class present in the canvas. This list is displayed to the user, who can then select the desired suggestions to incorporate into the initial model.

It is worth noting that, while conducting the user study we asked the participants to complete a free task where they have the flexibility to toggle between different modes or choose not to select any specific mode (**No Assistance Mode**).

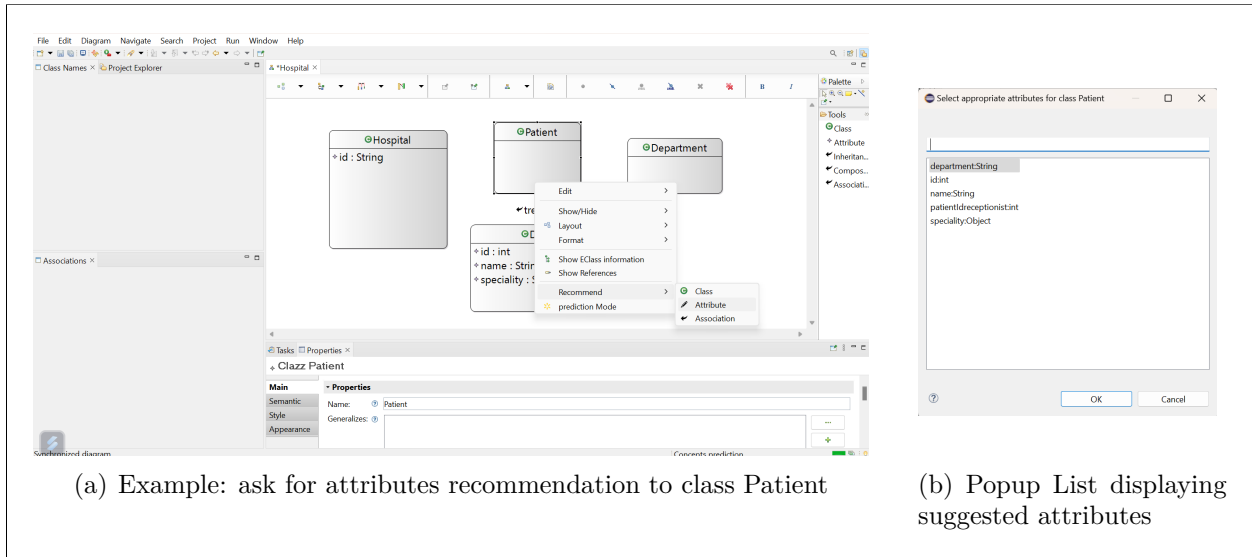


Fig. 5.2. Implementation Snapshot: On request Mode

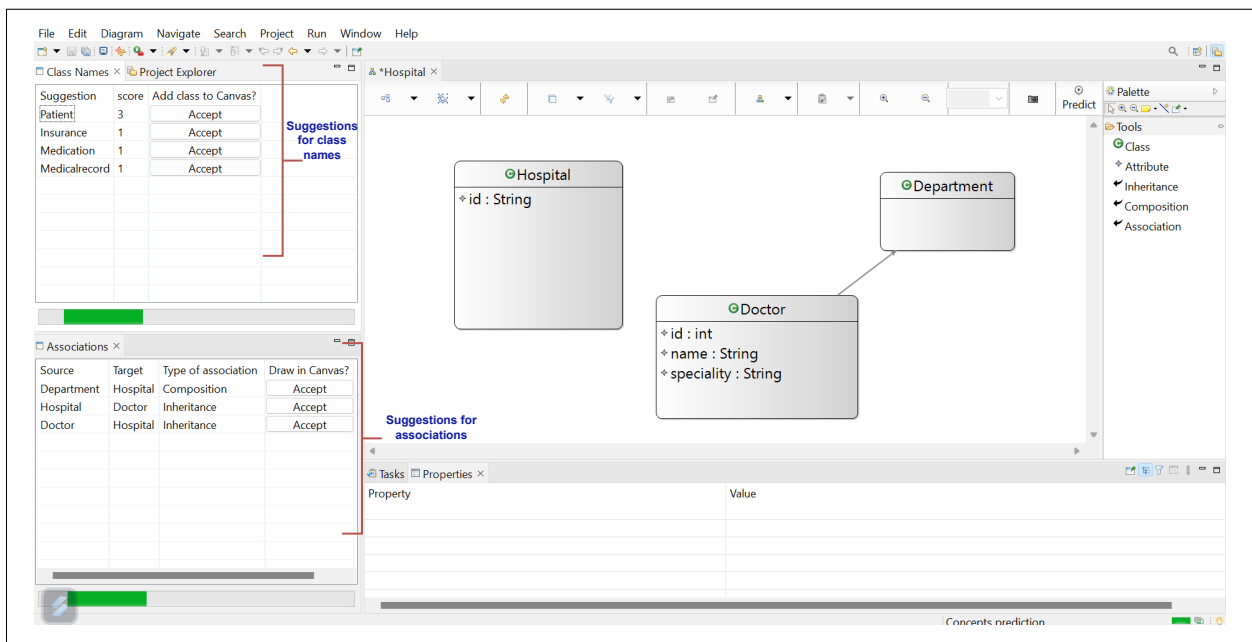


Fig. 5.3. Implementation Snapshot: Automatic Mode

5.1.3. Implementation details

During the development of the editor, we focused on several implementation details and aspects to ensure a robust and user-friendly experience.

- The presentation of recommendations and suggestions to the user was a key focus. We implemented strategies to prioritize frequently suggested concepts, as they were deemed more relevant. This ensured that the most valuable and contextually appropriate recommendations were presented to the user.

- To improve performance and efficiency, a caching system was implemented. We employed various techniques, including the use of threads, hash maps, and design patterns such as singleton and observer. This caching system helped in storing and retrieving frequently used data, resulting into optimizing the response time of the editor. Additionally, jobs were initiated to predict related class names and attributes for existing classes in the canvas, as well as related associations when a new class was added.
- Instant refresh capability was integrated into the editor to provide users with a smoother and more responsive experience. This allowed for real-time updates and immediate feedback when adding or modifying elements in the class diagram.
- The design of the editor was carefully considered to ensure an intuitive and visually appealing interface. Special attention was given to the layout, color style, and general user experience.

5.2. Evaluating the usefulness of Modeling Assistance

From the user’s perspective, there is a lack of understanding of the strengths and weaknesses of various types of assistants, which are constructed with different techniques and designed to assist with different tasks. Given the prominent role that modeling assistants are expected to play within MDE tools [32, 31], this understanding is essential for future modeling assistant developers. In our work we conduct a user study to fill this void and evaluate the usefulness of the proposed modeling assistance approach.

Other researchers have previously used interviews to learn more about problems with MDE tool adoption in industry [51]. We don’t use an interview-based method since we’re interested in quantifying the benefits of including an assistance into the modeling process, and that can only be done with actual users of the tool.

5.2.1. Aim and scope

The goal of this study is to understand how few-shot learning can improve the measured and perceived utility of model completion mechanisms.

That is, in terms of the Goal-Question-Metric perspectives [9]:

<i>Purpose</i>	Observe and analyze the perceived utility of few-shot learning on model completion recommendations from the end-user’s point of view.
<i>Issue</i>	
<i>Object</i>	
<i>Context</i>	
<i>Viewpoint</i>	
<i>Question</i>	Quality of recommendation
<i>Metrics</i>	Objective and quantitative: correctness (recall)
<i>Question</i>	User behavior
<i>Metrics</i>	Objective and quantitative: time to complete, creativity (divergence from mean editing profiles), etc
<i>Question</i>	User-perceived utility
<i>Metrics</i>	Subjective and qualitative: usefulness of recommender mechanisms, confidence in editing, etc

Tableau 5.1. The goal of the user study.

5.2.2. Research questions

Based on the goals, we define the following research questions.

- RQ1.** *What is the impact of incorporating few-shot learning techniques for suggesting modeling elements on the **time** required to complete diagrams? And which suggestion mode proves most effective in optimizing diagram completion time?*
- RQ2.** *Does few-shot learning for suggesting modeling elements reduce solution **diversity**? If yes, which suggestion mode has the most significant impact in this regard?*
- RQ3.** *How does the utilization of few-shot learning for suggesting modeling elements contribute to enhancing the **efficiency** of completing modeling tasks, as measured by acceptance rate and assessment rate? Which suggestion mode yields the most favorable impact in this context?*
- RQ4.** *How does few-shot learning affect the **modeling experience**, as perceived by the users?*

5.2.3. Experimental setup

We follow a **within subject design** where the same group of participants is exposed to various experimental setups or treatments, and their responses or performance are measured and compared across these conditions. We hire 15 participants, each group of 5 will perform three different modeling tasks as explained in table 5.2. We make sure that the three groups

test the three different modes and report the results. The participants are selected based on their familiarity with object-oriented modeling and the use of similar tools. We note that we don't consider the gender in the selection criteria for participants in this user study. We just gather diverse perspectives to ensure that the editor is effective and usable for a wide range of users.

5.2.3.1. Tasks: Recommendation alternatives.

- (1) No assistance. The designer is tasked with completing a modeling task without the support of any recommendation mechanism.
- (2) Recommendation On Request. (**RR**) The designer asks for a specific type of prediction (Looks for potential attributes, concepts or associations)
- (3) Automatic Recommendation (on Triggers) (**AR**); While modeling, and as things progress, concepts and operations are suggested. Adding elements to the canvas triggers the suggestion of elements that the implemented algorithm finds relevant.
- (4) Recommendation at the End (**FR**): Evaluate and suggest what is missing at the end.

5.2.3.2. Modeled domains.

Three diverse domain models of similar sizes are selected, encompassing multiple disciplines. We present them to the participants with comprehensive descriptions (Meta stories); details are in the annex .3. Participants then are tasked with designing class diagrams using the implemented proof-of-concept. One other Domain model, Hospital Management System, is used as an explanatory example in a demo that we sent to each participant before the experiment session starts and we asked him to visualize it (Annex A. .1).

- D1: Class Diagram for Online Shopping System
- D2: Class Diagram for Banking System
- D3: Class Diagram for Hotel Management System

		Domain			<i>Sample size</i>
		D1	D2	D3	
3* Recommender mechanism	RR	G1 (n = 5)	G2 (n = 5)	G3 (n = 5)	15
	AR	G2 (n = 5)	G3 (n = 5)	G1 (n = 5)	15
	FR	G3 (n = 5)	G1 (n = 5)	G2 (n = 5)	15
<i>Sample size</i>		15	15	15	

Tableau 5.2. Experimental design

5.2.3.3. Variables.

We define in this section variables by which we answer the research questions.

- **RQ1: Time to complete task**

Our objective is to precisely measure the amount of time required for each participant to complete the modeling task. To achieve this, we employ a logging system that tracks all user operations and decisions, along with the corresponding timestamps of when they occurred. Our focus is then directed towards contrasting the time discrepancies between different modes. (**Q:** Which mode empowers participants to complete the task more efficiently?)

- **RQ2: Diversity between completed diagrams**

Our objective is to assess how the recommendation tool affects the creativity of participants. We do this by comparing the sequences of elements added to the model under construction by participants (concepts and attributes) and evaluating the degree to which these sequences differ. By examining these differences, we can ascertain the participants' level of creativity and determine whether the tool is restricting or enhancing their diverse and creative output.

We apply the following formula 5.2.1 to calculate the diversity between two models, modelSource M_i and modelTarget M'_i that answer the same task and are built using the same mode:

$$\text{Distance} = 1 - \frac{\text{inter}(M_i, M'_i)}{\text{Union}(M_i, M'_i)} \quad (5.2.1)$$

The calculated distance and diversity are inversely proportional to one another. When the calculated distance is closer to 0 it indicates that the models being compared are extremely similar and share a significant number of their elements. Therefore, when the calculated distance approaches 1, it indicates that the models are extremely dissimilar and share few features. This suggests that the participants have added unique elements to each model, resulting in a greater degree of diversity between the models.

- **RQ3: Efficiency of suggesting modeling elements on modeling task**

- **Acceptance Rate:**

This metric measures the extent to which the suggested concepts are accepted by the users. The formula used for calculating it is as follows:

$$\text{Acceptance Rate} = \frac{\text{Accepted concepts from suggestions}}{\text{Total number of suggested concepts}} \quad (5.2.2)$$

A higher acceptance rate indicates that a larger proportion of the suggested concepts are valuable and are incorporated into the final design, whereas a lower acceptance rate means that fewer suggestions are accepted. We may assess the efficiency and utility of the suggested elements (we consider here class names) in the design process by looking at the acceptance rate.

- **Assessment Rate:**

This metric measures the proportion of accepted concepts from the suggestions

compared to the total number of concepts in the model. The formula for calculating the Assessment Rate is as follows (5.2.3):

$$\text{Assessment Rate} = \frac{\text{Accepted concepts from suggestions}}{\text{Total number of concepts in model}} \quad (5.2.3)$$

This metric provides insights into the effectiveness of the suggested concepts in improving the overall quality of the model. A higher assessment rate indicates that a larger proportion of the suggested concepts are accepted and incorporated into the model, reflecting a more successful integration of suggestions. A lower assessment rate, on the other hand, indicates that fewer suggestions are considered relevant or important for the model.

We note that, the acceptance rate concentrates on the acceptance of specific suggested concepts, whereas the assessment rate provides a more general evaluation of the impact and effectiveness of the suggestions on the overall model. Both metrics provide valuable insights into the utility and success of implementing suggestions, though from distinct perspectives.

- RQ4: **Subjective and qualitative measures: Modeling experience**

We ask every participant to answer some questions after he/she finishes the tasks; We use Likert scale³, a psychometric scale commonly involved in research that employs questionnaires, to evaluate the obtained results.

We note that, each question on the forms has a negative column to make sure that users are reading the questions carefully and providing accurate answers rather than choosing answers at random.

5.3. Quantitative Analysis

5.3.1. Time to complete a task

We present in table 5.3 the mean times needed to complete the tasks using various modes for drawing the domain models.

Mode	Time Spent
No assistance	00:14:27
On Request	00:11:16
On Trigger	00:10:46
Assess At End	00:17:03

Tableau 5.3. Results evaluating the time spent for each mode

³https://en.wikipedia.org/wiki/Likert_scale

From the mean times taken for each mode, we can make some initial observations. We note that the standard deviation of these values is relatively low, indicating then a small amount of variability in the data.

- The "no Assistance" mode has the longest mean time of 14:27, indicating that participants took the most time to complete the tasks without any assistance or recommendations.
- In comparison to the "noAssistance" mode, the "On Request" mode has a shorter mean time. This shows that participants were able to finish the tasks more quickly when they had the option to request certain advice or assistance.
- The "On Triggers" or "automatic " mode has the quickest mean time of all the settings. This suggests that when the system made suggestions in response to the addition of elements into the canvas, participants were able to complete the tasks more quickly.
- The results support the initial hypothesis that the "At End" mode does indeed have the longest average duration. The additional assessment and decision-making that occurs after the user thinks their job is finished explains the prolonged length.

Statistical test

While comparing means can provide a basic understanding of group differences, it may not offer a comprehensive and rigorous analysis of the data and because we are dealing with small groups, it becomes difficult to assume a normal distribution. Hence, we use a non-parametric statistical tests to address this issue.

One commonly employed non-parametric test is the Kruskal-Wallis test [19], which compares the distributions of multiple independent groups or samples. This test allows us to assess the significance of observed differences between groups without relying on assumptions of normality.

Modes	sig
On Request vs On Trigger	0.616
On Request vs At End	0.022
On Request vs No Assistance	0.006
On Trigger vs At End	0.014
On Trigger vs No Assistance	0.035
At End vs No Assistance	0.455

Tableau 5.4. Pairwise comparison of treatment- Time To complete the task

- Completing the modeling task was notably faster, in the "On Request" mode compared to the "At End" mode (significance = 0.022) and the "No Assistance" mode (significance = 0.006). This suggests that having the ability to request assistance helped in creating diagrams.

- The "On Trigger" mode demonstrated significantly faster completion times compared to the "At End" mode (sig = 0.014) and the "No Assistance" mode (sig = 0.035), reinforcing that automatic suggestions based on user actions contributed to efficiency.
- However, we note no significant difference in completion times between the "At End" mode and the "No Assistance" mode (sig = 0.455).

Key Findings

- Our results show that providing users with real-time suggestions and support during their modeling activities lead to a faster completion of the diagram.

5.3.2. Diversity of solutions

We Compare the obtained models against each others as explained in section 5.2.3.3.

Mode	Diversity	std
No Assistance	0.76	0.1
On Request	0.67	0.15
On Trigger	0.70	0.09
Assess At End	0.74	0.12

Tableau 5.5. Results evaluating the diversity between obtained models cross different modes

- The obtained models have a diversity score of 0.76 in the "No Assistance" mode, which denotes a rather high level of variance in the class diagrams that were constructed. This suggests that the models differ from one another to a significant degree, reflecting diverse design choices made by the participants. The standard deviation of 0.1 indicates a moderate level of dispersion around the mean.
- The diversity score drops to 0.67 in the "On Request" mode, indicating a less variation than in the "No Assistance" mode. The models are still diverse from one another even though the diversity score is lower. However, the higher standard deviation of 0.15 suggests a greater dispersion of results around the mean.
- The "On Trigger" mode exhibits a diversity score of 0.70, indicating a moderate level of variation in the generated models. While the models are not as diverse as in the "No Assistance" mode, they still differ significantly from one another. The lower standard deviation of 0.09 suggests less dispersion around the mean, indicating relatively consistent results.
- The diversity score rises to 0.74 in the "Assess At End" mode, showing more variation than in the "On Trigger" mode. Although not as much as in the "No Assistance" option, the models in this mode do differ from one another substantially. The resulting

values' variability is suggested by the standard deviation of 0.12, which points to a modest level of dispersion around the mean value.

Statistical test We conduct the Kruskal-Wallis statistical test, as elaborated earlier, to analyze the data.

Modes	sig
On Request vs On Trigger	0.636
On Request vs At End	0.051
On Request vs No Assistance	0.007
On Trigger vs At End	0.0140
On Trigger vs No Assistance	0.025
At End vs No Assistance	0.443

Tableau 5.6. Pairwise comparison of treatment- Diversity

The "Modes" column in table 5.6 represents the pair of tasks being compared, and the "sig" column indicates the significance level associated with each comparison.

- (1) On Request vs On Trigger: No significant difference in diversity ($p = .636$) which means that regardless of whether the designer actively seeks recommendations or relies on suggestions triggered by her/his actions, the resulting models exhibit similar levels of variation and uniqueness.
- (2) On Request vs At End: Marginally significant difference in diversity ($p = .051$). It implies that the approach of requesting recommendations during the design process and accepting suggestions into the canvas leads to a slightly different set of models compared to evaluating missing elements at the end.
- (3) On Trigger vs At End: Significant difference in diversity ($p = .014$). This indicates that models generated with on-the-go suggestions may differ less in terms of their range and variety compared to those generated by evaluating missing elements at the end.
- (4) On Request vs No Assistance and On Trigger vs No Assistance: Significant difference in diversity (respectively; $p = .007$, $p = .025$). This indicates that the presence of recommendations significantly impacts the range and variety of the models generated by the system.
- (5) At End vs No Assistance: the obtained models do not significantly differ in terms of their diversity. This implies that the absence of recommendations does not significantly impact the range and variety of the models generated by the designer. Evaluating missing elements at the end without assistance still results in a diverse set of models, similar to when no assistance is used at all.

Key Findings

- Our results show that using assistance lead to obtaining less diverse models.

5.3.3. Contribution of suggestions

5.3.3.1. Acceptance Rate (concepts).

Mode	Acceptance Rate
On Request	0.34
On Trigger	0.36
Assess At End	0.29

Tableau 5.7. Results evaluating the Acceptance rate for each mode

The acceptance rates provide insights into the users' willingness to accept the suggestions offered by each mode. A higher acceptance rate indicates that the suggestions are considered useful and valuable by the users.

- The "On Trigger" mode appears to have the highest acceptance rate based on these statistics, indicating that the dynamic and contextual nature of this mode allows the suggestions to align closely with the user's design intentions, leading to a higher acceptance rate. As the user adds elements to the canvas, the tool triggers relevant suggestions, effectively aiding the user's perception and decision-making during the design process. The "On Request" mode exhibits also a reasonable acceptance rate, indicating that users are open to accepting a significant portion of the recommended concepts when they actively seek suggestions. The "Assess At End" mode shows a lower acceptance rate. A way to explain this is that users tend to view suggestions introduced at the end of their design process as an indication of imperfections in their initial design. This perception of completion and reluctance to make further changes contribute to the lower acceptance rate for end-provided suggestions.

Modes	sig
On Trigger vs at End	<.001
On Request vs at End	<.001
On Request vs On Trigger	.88

Tableau 5.8. Pairwise comparison of treatment- Acceptance Rate

We preform the Kruskal-Wallis statistical test as explained previously, and based on these results in table 5.10, it can be concluded that the acceptance rates differ significantly

between the "On Request" and "At End" modes, as well as between the "On Trigger" and "At End" modes. However, there is no significant difference in the acceptance rates between the "On Request" and "On Trigger" modes.

5.3.3.2. Assessment Rate (concepts).

Mode	Assessment Rate
On Request	0.43
On Trigger	0.48
Assess At End	0.2

Tableau 5.9. Results evaluating the Assessment rate for each mode

- The table 5.9 indicates that the "On Trigger" mode has a higher acceptance rate for the suggested concepts (48%) compared to the "On Request" mode (43%). This means that the suggestions generated during the modeling process were well-received and beneficial for the modeling task. In contrast, the "Assess At End" mode has the lowest assessment rate of 20%. This indicates that the suggestions provided at the end of the design process are less accepted. It suggests that users were less likely to implement suggestions at the conclusion, possibly due to a sense of completion and reluctance to make further modifications.

The Kruskal-Wallis statistical test performed on these obtained values shows again that here is no significant difference in the assessment rates between the "On Request" and "On Trigger" modes in contrast to the other mode comparisons.

Statistical test

Modes	sig
On Trigger vs at End	<.001
On Request vs at End	<.001
On Request vs On Trigger	.65

Tableau 5.10. Pairwise comparison of treatment- Assesment Rate

Key Findings

- Our results show that particularly on request and on trigger modes, are effective in providing suggestions that are accepted and beneficial to users during the design process.

5.4. Qualitative Analysis

The participants were asked to rate different aspects on a scale from 1 to 5. To present the results in a clear and visually appealing manner, we use bar plots, where the intensity of the scores was represented using a color gradient. High scores are represented by an intense red color, indicating a strong presence of the associated aspect. In contrast, low scores are represented by a lighter shade of red, indicating a weakened presence of the aspect.

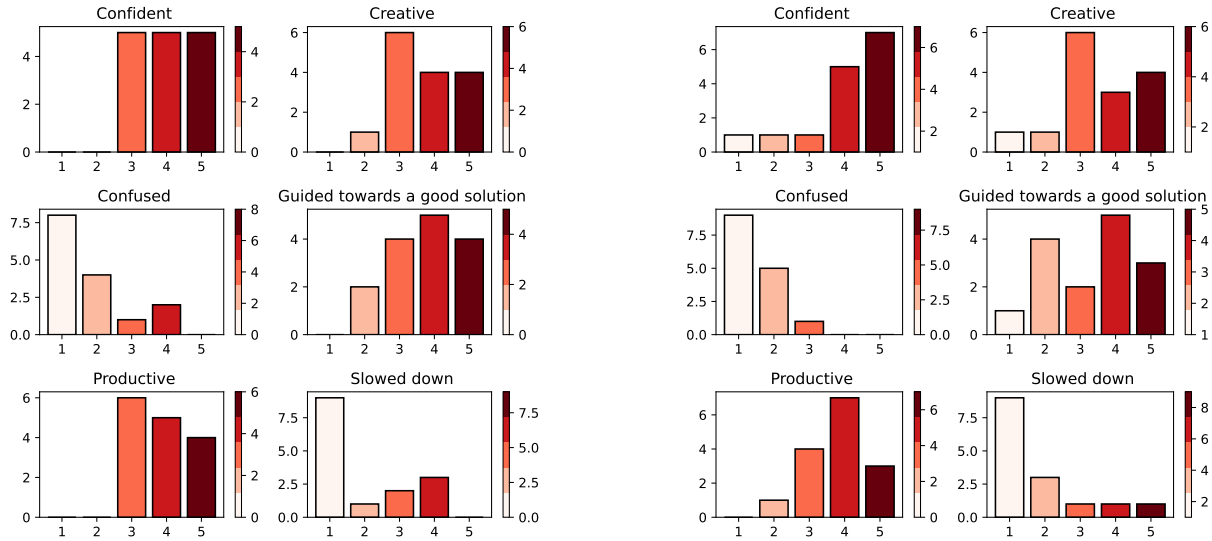
5.4.1. Results during experiments

The aim of the first question is to investigate the psychological impact of having assistance mode alongside the designers. Participants were asked to rate their feelings during the experiment using a scale of 1 to 5 for various emotions: Confident, Creative, Confused, Guided towards a good solution, Productive, and Slowed down.

Overall, the results showed positive feedback. Figure 5.4 presents a bar plot depicting the ratings provided by participants for each emotion and each mode, using a color gradient to indicate the intensity of the scores.

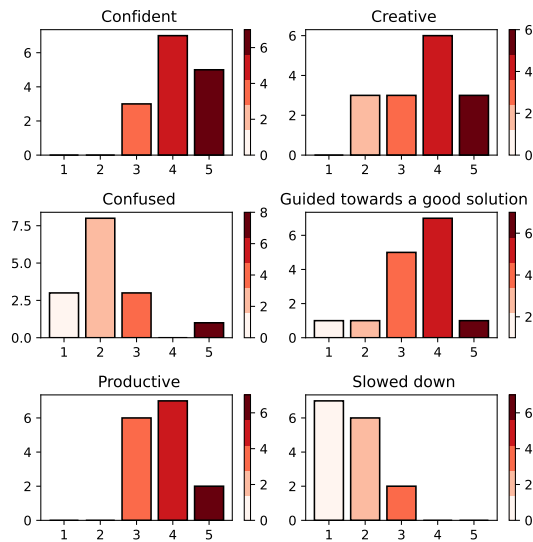
We note that, most of the participants said that they found the *at end* mode, where they receive suggestions as a final step of the modeling process, less confusing compared to the rest of the modes. This can be explained by the increased cognitive load caused by the addition of suggestions during the modeling process. In fact, harmonizing their own design decisions with the received suggestions may require additional mental effort, which may result in confusion or decision fatigue.

However, results show that in *automatic mode* where suggestions appear while modeling, participants weren't feeling slowed down. In other words, when suggestions are accurately aligned with participants' modeling goals, they can enhance the modeling process by providing valuable direction, leading to a smooth modeling experience without significant interruptions or delays.



(a) OnRequest Mode

(b) At End Mode



(c) Automatic Mode

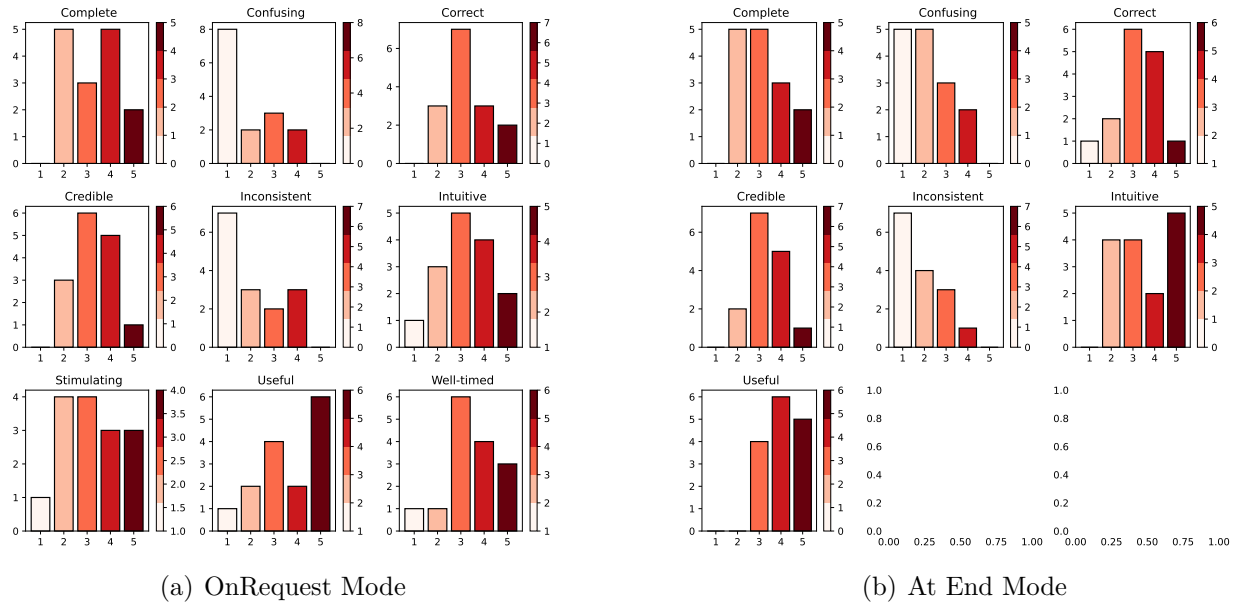
Fig. 5.4. Answers to question: During the experiment, I felt...

5.4.2. Effectiveness of the recommendations

The second question in the survey was about the recommendations provided by the software. Using the Likert scale, participants provided scores to indicate their level of agreement or disagreement with each concept in relation to the recommendations received. The list of the concepts is as follows: [Complete, Confusing, Correct, Credible, Inconsistent, Intuitive, Useful]. The two adjectives, [‘Well timed and Simulating recommendations’] are absent when

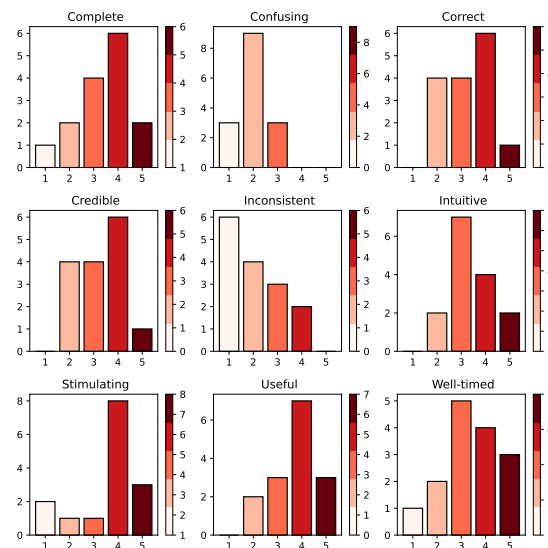
it is about the at-end mode because they are not relevant in that situation; recommendations are provided as a final step in the modeling process. Again, we include some negative adjectives to make sure participants are not answering randomly and there's coherence in their answers.

As shown in figure 5.5, most of the participants rate the "usefulness" of the recommendations highly across all modes. Additionally, both adjectives "confusing" and "inconsistent" received relatively low ratings from participants in the evaluation.



(a) OnRequest Mode

(b) At End Mode



(c) Automatic Mode

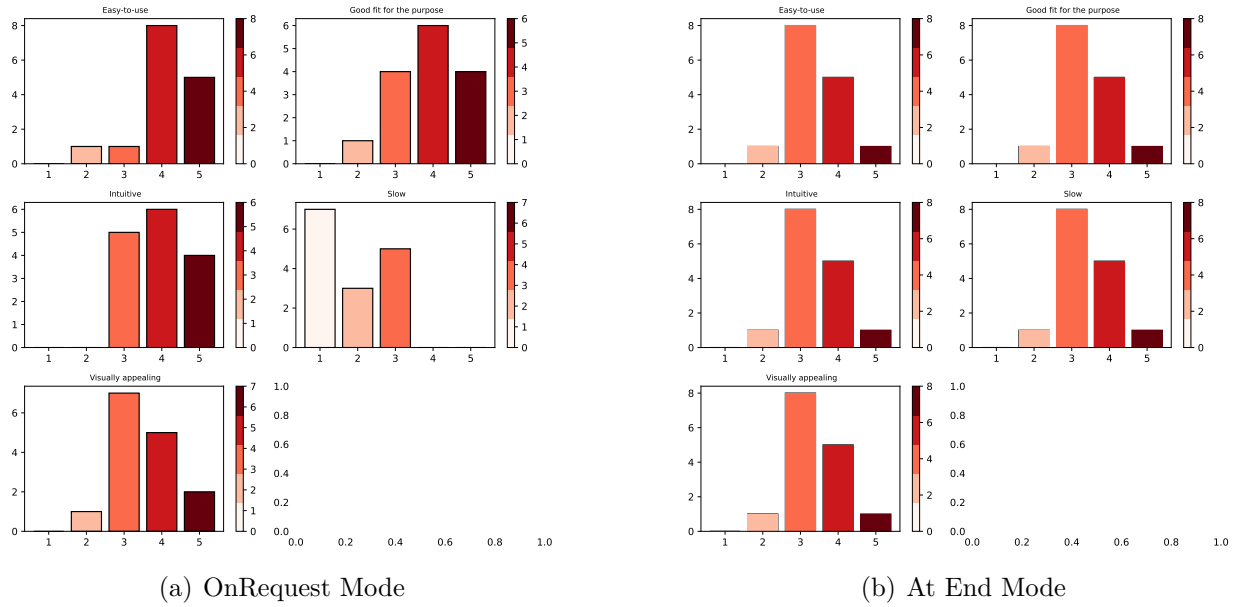
Fig. 5.5. Answers to question: I found the recommendations...

5.4.3. Impact of the tool

The final question in the form was related to the tool itself. Users were asked to report their feelings about the tool rating the following adjectives: 'Easy-to-use', 'Good fit for the purpose', 'Intuitive', 'Slow', 'Visually appealing'.

The positive ratings for most of adjectives such as 'Easy-to-use' and 'Intuitive' across all modes, reinforce the notion that participants had a positive overall impression of the tool.

However, figure 5.6 show that the *at end* mode is perceived as slower compared to the other modes. In fact, in the other modes, recommendations are generated and displayed in real-time or as the user advances through the modeling task. In these modes, we employ a caching system in which calculations and recommendations are stored and readily accessible for immediate display, thereby reducing any perceived delay. In contrast, the "At End" mode lack this caching system.



(c) Automatic Mode

Fig. 5.6. Answers to question: I found the tool....

5.4.4. Participants feedback

It would be valuable to analyze any other direct feedback provided by participants to gain further insights and contextualize the previous ratings. We list some of the opinions of the subjects expressed in the open-ended questions of the Questionnaire in the annex .2.

- Which features would you improve, and how?
 - Very cool tool! Thank you!! It's just a little bit slow at the end, so maybe it's better to show a message to the user telling him it's processing the results.
 - When recommendations increase, it's difficult to choose from them. Perhaps filtering some out? Also, I can't navigate through the recommendations, can't sort or search which slows me down.
 - Include user feedback on the suggestions (the model may suggest a class/attribute that I already added but with another name/form).
- What features did you miss?
 - I think conceptually, the suggestions are good, but in certain cases, there were suggested classes that I would put as attributes instead. This is a design decision, of course, but maybe you can specify that the tool is meant to design models with more classes and attributes (?).
 - It would be good to disregard suggestions. And having attribute suggestions would also be helpful. Or refactoring suggestions.
- Do you have any other remarks?
 - Very useful and helpful tool
 - I really liked how the association arrow flips; the suggestions are simulating.
 - The list of recommendation gets long, so after some time it becomes difficult to use, otherwise, it's a great tool!
 - I enjoyed the tool! It makes designing UML faster.

Key Findings

Based on the feedback, it can be concluded that the assistance tool for model completion has a positive impact on the modeling process. By providing real-time suggestions and guidance, this approach can assist software modelers in improving their productivity, accuracy, and overall modeling outcomes. However, there is room for improvement in terms of performance, recommendation management, and the inclusion of additional features. These insights can guide further development and refinement of the tool to enhance its overall effectiveness and user satisfaction.

5.5. Threats to validity

In this section, we analyse the threats to the validity of our study.

- (1) **Domain-Specific Bias:** The choice of domain models may introduce domain-specific biases. The findings may be specific to these domains and may not fully

capture the tool’s effectiveness in other domains.

=> We already include a broader range of domains to mitigate to this.

- (2) **Limited Sample Size:** The experiment involves a relatively small sample size, consisting of fifteen participants divided into three groups. A limited sample size can reduce the statistical power of an analysis and its generalizability.

=> To mitigate this threat, we carefully select participants and we ensure that they are representative of the target population. Also, since we are considering a within-subject design where participants are exposed to multiple conditions or treatments, we reduce the variability across participants and allow for more robust comparisons within the limited sample size. Additionally, we consider the replication of this study in the near future to validate and strengthen these initial findings.

- (3) **Learning Effect:** The experiment involves multiple tasks performed by the same group of participants. There is the prospect of a learning effect, in which participants may improve their performance or become more familiar with the instrument over time, potentially influencing their responses and skewing the results..

=> To mitigate this threat, we randomize the order of modes across participants to minimize the influence of order effects and potential biases, allowing for more reliable and valid results.

- (4) **Statistical corrections:** One important point to consider is the absence of statistical corrections in our analysis. While the Kruskal-Wallis test provides valuable insights into the differences among assistance modes and their impacts on not only task completion times but also the diversity of the obtained diagrams and the efficiency of the suggestions, it does not inherently account for multiple comparisons or potential false positives that could arise from conducting numerous tests without adjustments. As a result, the risk of Type I errors, or false positives, may be heightened due to the lack of correction procedures. Therefore it’s crucial to interpret our findings and acknowledge that further research or additional analyses may be necessary to validate and refine our conclusions.

5.6. Conclusion

In this chapter, we have delved into the study of the usefulness of prompt-based modeling assistance for software modelers during the design task. We introduced a graphical editor as a platform to implement and evaluate the prompt-based approach, providing a user-friendly interface for participants to interact with the modeling assistance system. Through a carefully designed user study, we gathered valuable feedback from participants regarding their perception of the utility of the proposed methodology. Indeed, this methodology has

the potential to enhance the modeling experience for software modelers, as validated by the positive feedback from participants.

Chapter 6

Conclusion and future work

6.1. Summary

In this thesis, we propose a novel approach based on few-shot prompt learning to enable large language models to solve completion tasks in modeling activities. We reformulate model completion as a semantic mapping problem that consists, firstly, in transforming modeling formalism elements into meaningful patterns of sequences of tokens to create prompts with learning shots. Then, we exploit the ability of LLMs to complete partial sequences following the specified patterns to recover elements that can be used for the completion. Those elements are transformed into constructs conforming to the modeling language syntax and suggested to the modelers. Although many research contributions were proposed to solve model completion problems, we do believe that none of them can be effectively used in a real setting, because of the resources needed, i.e., large training datasets, and the limited performance they offer. Initially, we explored another path which consisted of leveraging the code data and using it to predict related elements and complete models through reverse engineering. However, despite our efforts, this approach proved to be a failure.

Besides, we do believe that the new proposed approach can be effective when modeling both static and dynamic diagrams for two main reasons. Firstly, it does not require to pre-train or fine tune language models on specific tasks or domain. Secondly, the used LLMs are trained on a huge volume of data, which makes it generalizable to many domains and different concept natures and relationships.

6.2. Future work

Although our approach shows promising results, it is still a first attempt and there is room for improvement.

Indeed, when defining a prompt, the elements of the already-defined partial diagrams have a great influence on the accuracy of the suggested token. A *calibration study* is

still necessary to determine the boundaries of the provided existing context to have the best suggestions. For example, when we added systematically the package name in the pattern, the results improved considerably, but we cannot determine whether this observation is valid only on the used benchmark.

Another consideration that has to be studied is the use of non-natural language elements such as symbols and digits. In our experiments, their existence generated poor results as these elements are rarely present in the data used for training LLM. We believe that a more sophisticated mapping of those elements would considerably improve the results.

In our approach, we use the advanced capabilities of GPT-3 by OpenAI, a state-of-the-art language model. However, this makes us rely on third-parties, which could limited the availability of the service or deny our access to it in the future. In such case, we should replace GPT-3 with a similar LLM.

Another aspect of future work involves expanding the scope of the user study. While our initial one involved only 15 participants based in Montreal, we acknowledge the need to replicate the study with a larger and more diverse group. By increasing the sample size and incorporating participants from a different background, we can obtain a more comprehensive understanding of the strengths and limitations of our approach. This would enable us to gather a broader range of feedback and insights, leading to more robust conclusions and recommendations.

Finally, we intend to explore different modeling formalisms expanding the range of completion tasks that can be tackled using our approach. We plan to expand our proof of concept to handle these new formalisms.

Overall, these future work seek to strengthen the empirical validation of our approach, collect more diverse feedback, and refine the implemented software for model completion to make it even more effective and adaptable across a broad range of modeling tasks.

Références bibliographiques

- [1] Eclipse sirius best practices: Version 5.1. <https://www.obeodesigner.com/en/best-practices>. [Online; 2023-05-22].
- [2] Modisco. <http://www.eclipse.org/gmt/modisco/>.
- [3] UML activity diagram examples. <https://www.uml-diagrams.org/activity-diagrams-examples.html>. [Online; accessed 12-October-2022].
- [4] Eclipse, “sirius,” 2014.
- [5] Miltiadis ALLAMANIS et Charles SUTTON : Mining Source Code Repositories at Massive Scale using Language Modeling. In *The 10th Working Conference on Mining Software Repositories*, pages 207–216. IEEE, 2013.
- [6] Chetan ARORA, Mehrdad SABETZADEH, Lionel BRIAND et Frank ZIMMER : 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*, pages 250–260, 2016.
- [7] Paul BAKER, Shiou LOH et Frank WEIL : Model-driven engineering in a large industrial context—motorola case study. In *Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005. Proceedings 8*, pages 476–491. Springer, 2005.
- [8] Gabriel BARBIER, Hugo BRUNELIERE, Frédéric JOUAULT, Yves LENNON et Frédéric MADIOT : *MoDisco, a Model-Driven Platform to Support Real Legacy Modernization Use Cases*, pages 365–400. 12 2010.
- [9] Victor R. BASILI, Gianluigi CALDIERA et H. Dieter ROMBACH : The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, volume 2, pages 528–532. Wiley, 1994.
- [10] Daniela BERARDI, Diego CALVANESE et Giuseppe DE GIACOMO : Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1):70–118, 2005.
- [11] Marco BRAMBILLA, Jordi CABOT et Manuel WIMMER : *Model-Driven Software Engineering in Practice: Second Edition*. Morgan amp; Claypool Publishers, 2nd édition, 2017.
- [12] Leo BREIMAN : Random forests. *Machine learning*, 45:5–32, 2001.
- [13] Tom B. BROWN, Benjamin MANN, Nick RYDER, Melanie SUBBIAH, Jared KAPLAN, Prafulla DHARWAL, Arvind NEELAKANTAN, Pranav SHYAM, Girish SASTRY, Amanda ASKELL, Sandhini AGARWAL, Ariel HERBERT-VOSS, Gretchen KRUEGER, Tom HENIGHAN, Rewon CHILD, Aditya RAMESH, Daniel M. ZIEGLER, Jeffrey WU, Clemens WINTER, Christopher HESSE, Mark CHEN, Eric SIGLER, Mateusz LITWIN, Scott GRAY, Benjamin CHESS, Jack CLARK, Christopher BERNER, Sam MCCANDLISH, Alec RADFORD, Ilya SUTSKEVER et Dario AMODEI : Language models are few-shot learners. In Hugo LAROCHELLE, Marc’Aurelio RANZATO, Raia HADSELL, Maria-Florina BALCAN et Hsuan-Tien LIN, éditeurs

- : *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [14] Hugo BRUNELIERE, Jordi CABOT, Grégoire DUPÉ et Frédéric MADIOT : Modisco: a model driven reverse engineering framework. *Information and Software Technology*, 56, 08 2014.
- [15] Lola BURGUEÑO, Robert CLARISÓ, Sébastien GÉRARD, Shuai LI et Jordi CABOT : An nlp-based architecture for the autocompletion of partial domain models. In Marcello La ROSA, Shazia W. SADIQ et Ernest TENIENTE, éditeurs : *Prof. of the 33rd International Conference on Advanced Information Systems Engineering (CAiSE 2021)*, volume 12751 de *Lecture Notes in Computer Science*, pages 91–106. Springer, 2021.
- [16] Thibaut CAPUANO, Houari A. SAHRAOUI, Benoît FRÉNEY et Benoît VANDEROSE : Learning from code repositories to recommend model classes. *Journal of Object Technology*, 21(3):3:1–11, 2022.
- [17] Meriem Ben CHAABEN, 2022 : https://github.com/meriembenchaaaben/Class_DiagramCompletionLeveragingCode.
- [18] Meriem Ben CHAABEN, Lola BURGUEÑO et Houari SAHRAOUI, 2023 : <http://hdl.handle.net/20.500.12004/1/C/ICSE/2023/001>.
- [19] WJ CONOVER : *Practical nonparametric statistics*. Wiley, 3rd édition, 1999.
- [20] Hoa Khanh DAM : Artificial intelligence for software engineering. *XRDS: Crossroads, The ACM Magazine for Students*, 25(3):34–37, 2019.
- [21] Davide DI RUSCIO, Ludovico IOVINO et Alfonso PIERANTONIO : What is needed for managing co-evolution in mde? In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*, pages 30–38, 2011.
- [22] Akil ELKAMEL, Mariem GZARA et Hanene BEN-ABDALLAH : An uml class recommender system for software design. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8, 2016.
- [23] Ahmed E. HASSAN : The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, pages 48–57, 2008.
- [24] José Antonio HERNÁNDEZ LÓPEZ, Javier Luis CÁNOVAS IZQUIERDO et Jesús SÁNCHEZ CUADRADO : Modelset: a dataset for machine learning in model-driven engineering. *Software and Systems Modeling*, 21(3):967–986, 2022.
- [25] Steven KELLY et Risto POHJONEN : Worst practices for domain-specific modeling. *IEEE Software*, 26(4):22–29, 2009.
- [26] JDMCK LEE et K TOUTANOVA : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [27] Pengfei LIU, Weizhe YUAN, Jinlan FU, Zhengbao JIANG, Hiroaki HAYASHI et Graham NEUBIG : Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, aug 2022. Just Accepted.
- [28] Senthil MANI, Vibha Singhal SINHA, Pankaj DHOOLIA et Saurabh SINHA : Automated support for repairing input-model faults. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 195–204, 2010.
- [29] Tomas MIKOLOV, Kai CHEN, Greg CORRADO et Jeffrey DEAN : Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [30] Parastoo MOHAGHEGHI, Wasif GILANI, Alin STEFANESCU et Miguel A FERNANDEZ : An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical software engineering*, 18(1):89–116, 2013.

- [31] Gunter MUSSBACHER, Benoit COMBEMALE, Silvia ABRAHÃO, Nelly BENCOMO, Loli BURGUEÑO, Gregor ENGELS, Jörg KIENZLE, Thomas KÜHN, Sébastien MOSSER, Houari SAHRAOUI et Martin WEYSSOW : Towards an assessment grid for intelligent modeling assistance. *In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.
- [32] Gunter MUSSBACHER, Benoit COMBEMALE, Jörg KIENZLE, Silvia ABRAHÃO, Hyacinth ALI, Nelly BENCOMO, Márton BÚR, Loli BURGUEÑO, Gregor ENGELS, Pierre JEANJEAN, Jean-Marc JÉZÉQUEL, Thomas KÜHN, Sébastien MOSSER, Houari SAHRAOUI, Eugene SYRIANI, Dániel VARRÓ et Martin WEYSSOW : Opportunities in intelligent modeling assistance. *Softw. Syst. Model.*, 19(5):1045–1053, sep 2020.
- [33] Fazle RABBI, Yngve LAMO, Ingrid YU et Lars KRISTENSEN : A diagrammatic approach to model completion. 09 2015.
- [34] Alec RADFORD et Karthik NARASIMHAN : Improving language understanding by generative pre-training. 2018.
- [35] Alec RADFORD, Jeff WU, Rewon CHILD, David LUAN, Dario AMODEI et Ilya SUTSKEVER : Language models are unsupervised multitask learners. 2019.
- [36] Martin ROBILLARD, Robert WALKER et Thomas ZIMMERMANN : Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.
- [37] Juri Di ROCCO, Claudio Di SIPIO, Davide Di RUSCIO et Phuong Thanh NGUYEN : A gnn-based recommender system to assist the specification of metamodels and models. *In 24th International Conference on Model Driven Engineering Languages and Systems (MODELS 2022)*, pages 70–81. IEEE, 2021.
- [38] James RUMBAUGH, Ivar JACOBSON et Grady BOOCH : *Unified Modeling Language Reference Manual, version 2.1.1*. OMG, 2007.
- [39] Rijul SAINI, Gunter MUSSBACHER, Jin L. C. GUO et Jörg KIENZLE : Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling*, 21(3): 1015–1045, 2022.
- [40] Victor SANH, Lysandre DEBUT, Julien CHAUMOND et Thomas WOLF : Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [41] Timo SCHICK et Hinrich SCHÜTZE : Few-shot text generation with natural language instructions. *In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 390–402, Online and Punta Cana, Dominican Republic, novembre 2021. Association for Computational Linguistics.
- [42] D.C. SCHMIDT : Guest editor’s introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.
- [43] Gehan M. K. SELIM, James R. CORDY et Juergen DINGEL : How is atl really used? language feature use in the atl zoo. *In 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 34–44, 2017.
- [44] Sagar SEN, Benoît BAUDRY et Doina PRECUP : Partial model completion in model driven engineering using constraint logic programming. 2007.
- [45] Taylor SHIN, Yasaman RAZEGHI, Robert L. LOGAN IV, Eric WALLACE et Sameer SINGH : AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online, novembre 2020. Association for Computational Linguistics.
- [46] Dave STEINBERG, Frank BUDINSKY, Ed MERKS et Marcelo PATERNOSTRO : *EMF: eclipse modeling framework*. Pearson Education, 2008.

- [47] Alexey SVYATKOVSKIY, Ying ZHAO, Shengyu FU et Neel SUNDARESAN : Pythia: Ai-assisted code completion system. *In 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining (KDD '19)*, September 2020.
- [48] Ashish VASWANI, Noam SHAZEER, Niki PARMAR, Jakob USZKOREIT, Llion JONES, Aidan N GOMEZ, Łukasz KAISER et Illia POLOSUKHIN : Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [49] Vladimir VIYOVIĆ, Mirjam MAKSIMOVIĆ et Branko PERISIĆ : Sirius: A rapid development of dsm graphical editor. *In IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238. IEEE, 2014.
- [50] Martin WEYSSOW, Houari A. SAHRAOUI et Eugene SYRIANI : Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling*, 21(3): 1071–1089, 2022.
- [51] Jon WHITTLE, John HUTCHINSON, Mark ROUNCFIELD, Håkan BURDEN et Rogardt HELDAL : A taxonomy of tool-related issues affecting the adoption of model-driven engineering. *Software and Systems Modeling*, 16(2):313–331, mai 2017.

Appendix A

.1. Video tutorial

Prior to their participation, a video was provided to ensure that all participants had the same level of knowledge about the tool and comprehended the tasks they were asked to complete during the user experience. This video served as a tutorial, outlining the tool's essential features, functionalities, and utilization instructions. It demonstrated step-by-step instructions for navigating the interface, gaining access to various features, and performing particular duties. By viewing the video, participants were able to familiarize themselves with the tool and comprehend the context of the subsequent user experience session. The video¹ can be accessed at [[this link](#)].

.2. Forms

In order to obtain qualitative results, we present the form that was administered to participants after each task as part of our data collection procedure. The purpose of the survey was to collect participants' valuable insights and subjective feedback regarding their experience with the task.

The questionnaire consisted of a Likert scale with multiple questions pertaining to various aspects of the task, including User properties, tool properties, and Recommendation properties. For the intended function, intuitiveness, velocity, and visual allure are crucial. On a scale from 1 to 5, with 1 representing the lowest rating and 5 the highest, participants were instructed to choose a numeric value to each aspect. The Likert scale provided a standard method for evaluating the subjective evaluations and preferences of participants.

¹<https://www.youtube.com/watch?v=5oUJJ0ial50list=PL1Srjx6J4Ps3NqubHpVdVCMXSGaQJwQBYindex=2>

Experiment questionnaire

Evaluation of the experiment

* Indicates required question

1. UserID *

2. Mode *

Mark only one oval.

- Automatic Suggestions Skip to question 9
- On request Suggestions Skip to question 3
- Completion at the end Skip to question 6

On Request Mode

On a scale of 1 to 5, where 1 is the lowest value and 5 the highest value, please, respond to the following questions.

3. During the experiment, I felt... *

Mark only one oval per row.

	1	2	3	4	5
Confident	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confused	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Guided towards a good solution	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Productive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slowed down	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. I found the tool... *

Mark only one oval per row.

	1	2	3	4	5
Easy-to-use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good fit for the purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Visually appealing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. I found the recommendations... *

Mark only one oval per row.

	1	2	3	4	5
Complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confusing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Credible	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Inconsistent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Stimulating (they gave me ideas I did not have)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Useful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Well-timed (they appeared right when I expected them)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Skip to question 12

Completion at end

On a scale of 1 to 5, where 1 is the lowest value and 5 the highest value, please, respond to the following question.

6. During the experiment, I felt... *

Mark only one oval per row.

	1	2	3	4	5
Confident	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confused	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Guided towards a good solution	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Productive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slowed down	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. I found the tool... *

Mark only one oval per row.

	1	2	3	4	5
Easy-to-use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good fit for the purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Visually appealing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. I found the recommendations.... *

Mark only one oval per row.

	1	2	3	4	5
Complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confusing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Credible	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Inconsistent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Useful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Skip to question 12

Automatic suggestions

On a scale of 1 to 5, where 1 is the lowest value and 5 the highest value, please, respond to the following question.

9. During the experiment, I felt... *

Mark only one oval per row.

	1	2	3	4	5
Confident	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confused	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Guided towards a good solution	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Productive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slowed down	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

10. I found the recommendations.... *

Mark only one oval per row.

	1	2	3	4	5
Complete	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Confusing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Correct	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Credible	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Inconsistent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Stimulating (they gave me ideas I didn't have)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Useful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Well-timed (they appeared right when I expected them)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. I found the tool... *

Mark only one oval per row.

	1	2	3	4	5
Easy-to-use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good fit for the purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Intuitive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Visually appealing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Skip to question 12

General

12. Which features would you improve, and how?

13. What features did you miss?

14. Do you have any other remarks?

.3. Tasks descriptions

- Banking System:

We need to model a banking system, which is a software application that manages financial transactions for a financial institution. It manages customer account information; communicates with various channels, such as ATMs; and processes transactions that can be either a withdrawal, a deposit, or a transfer. A bank can have different types of accounts. The system must maintain accurate and up-to-date information about each customer. Finally, the banking system also manages employee information.

- Hotel System:

We need to model a hotel system that is going to be used by hotels to manage their operations and provide services to their guests. The system should be able to handle various types of room reservations, such as single room, double room, suite, and so on. It should also be able to handle guest check-ins and check-outs. This involves collecting guest information, assigning rooms, and processing payments. Restaurant management is another important aspect of the hotel domain. It involves managing restaurant reservations, tracking table availability, and processing food and beverage orders. Additionally, the software should be able to schedule housekeeping tasks, track the status of cleaning and maintenance tasks, and manage housekeeping staff assignments. Finally, this system should be able to handle various types of staff tasks.

- Online shopping system:

We need to model an online shopping system for a software application that is going to be used to manage various operations and provide e-commerce services to customers. Product catalogs, including descriptions, images, pricing, and availability should be stored and managed by the software. The system must be able to process online orders, including tracking shipments, managing returns, and processing payments securely.

The system should be capable of scheduling deliveries for orders, managing the availability of products and resources, and providing customer support services.