Software and Systems Modeling (2025) 24:923–948
https://doi.org/10.1007/s10270-025-01263-8

SPECIAL SECTION PAPER

On the use of large language models in model-driven engineering

Juri Di Rocco[1] · Davide Di Ruscio[1] · Claudio Di Sipio[1] · Phuong T. Nguyen[1] · Riccardo Rubei[1]

Received: 20 March 2024 / Revised: 27 November 2024 / Accepted: 21 December 2024 / Published online: 31 January 2025

Abstract

Model-driven engineering (MDE) has seen significant advancements with the integration of machine learning (ML) and deep learning techniques. Building upon the groundwork of previous investigations, our study provides a concise overview of current large language models (LLMs) applications in MDE, emphasizing their role in automating tasks like model repository classification and developing advanced recommender systems. The paper also outlines the technical considerations for seamlessly integrating LLMs in MDE, offering a practical guide for researchers and practitioners. Looking forward, the paper proposes a focused research agenda for the future interplay of LLMs and MDE, identifying key challenges and opportunities. This concise roadmap envisions the deployment of LLM techniques to enhance the management, exploration, and evolution of modeling ecosystems. Moreover, we also discuss the adoption of LLMs in various domains by means of model-driven techniques and tools, i.e., MDE for supporting LLMs. By offering a compact exploration of LLMs in MDE, this paper contributes to the ongoing evolution of MDE practices, providing a forward-looking perspective on the transformative role of large language models in software engineering and model-driven practices.

Keywords LLMs · Generative AI · Model-Driven Engineering

1 Introduction

Model-driven engineering (MDE) promotes the adoption of models to allow for the specification, analysis, and promotion of complex software systems [77]. A modeling ecosystem is made of available models, transformations, code generators, and a plethora of software tools. With the integration of machine learning (ML) techniques, particularly in the context of modeling ecosystems, MDE has seen significant advancements lately [32].

Large language models (LLMs), such as GPT−3.5, have demonstrated remarkable proficiency in understanding and generating human-like text [64, 70, 75]. Very recently, there has been a dramatic increase in the number of applications of LLMs and pre-trained models (PTMs) in software engineering in general [69], and model-driven engineering in particular. To name but a few, LLMs have been widely used in testing [84], code generation [57], qualitative research [12], summarization [33], or commit message generation [93]. However, to the best of our knowledge, there exists no work to provide a panorama view of current applications of LLMs in MDE, as well as to sketch a roadmap for future research directions in the domain.

Our work has been conducted to fill such a gap, providing an overview of the existing applications of LLMs in the MDE domain. We acknowledge that many techniques used in AI-supported SE could potentially be adapted for MDE tasks. However, this paper specifically aims to investigate the unique challenges and methodologies that arise when integrating LLMs within the MDE paradigm. The primary reason for excluding papers related solely to AI-supported SE is to maintain a clear and concentrated scope on MDE-

Communicated by Javier Troya and Alfonso Pierantonio.

✉ Davide Di Ruscio
davide.diruscio@univaq.it
https://www.disim.univaq.it/DavideDiRuscio

Juri Di Rocco
juri.dirocco@univaq.it

Claudio Di Sipio
claudio.disipio@univaq.it

Phuong T. Nguyen
phuong.nguyen@univaq.it
https://www.disim.univaq.it/ThanhPhuong

Riccardo Rubei
riccardo.rubei@univaq.it

[1] Università degli studi dell'Aquila, L'Aquila, Italy

specific applications. Essentially, including SE papers would broaden the scope too much, thus shifting the focus. Moreover, while textual encodings of models (e.g., XML) provide a common ground for interoperability between SE and MDE, the context and objectives of their use in MDE are distinct. In MDE, these encodings are not merely textual representations but are used as inputs for model transformations, code generation, and other automated processes that are central to the MDE methodology. Therefore, we focused on studies that directly address these unique aspects of MDE. The discussion involves the utilization of LLMs for automated classification of model repositories and the development of advanced recommender systems within modeling ecosystems. This paper builds upon the foundations laid by our previously published work entitled "*Machine Learning for Managing Modeling Ecosystems: Techniques, Applications, and A Research Vision*" [32]. While the previous chapter explored the applications of traditional ML and deep learning (DL) in MDE, this paper goes one step forward, focusing on the utilization of LLMs to further enhance the capabilities of MDE.

To provide a comprehensive understanding, the paper goes deep into the technical intricacies of applying LLMs in MDE (LLM4MDE), as well as the other way round, i.e., using MDE to facilitate the adoption of LLMs (MDE4LLM). In particular, it outlines the specific steps and considerations necessary for effectively supporting MDE tasks by means of LLMs, and vice versa, i.e., promoting the inclusion of LLMs with the help of MDE. This ensures seamless synergy between language model understanding and the complexities of model-driven systems.

In particular, we aim at answering the following research questions:

- **RQ**$_1$: *How has existing research explored the application of LLMs in MDE tasks?* By conducting a systematic literature review (SLR) [49], we investigate *(i)* the extent and manner in which current studies integrate LLMs into MDE tasks (LLM4MDE); and *(ii)* the reverse scenario, where the MDE paradigm is employed to enhance the capabilities of LLMs (MDE4LLM);
- **RQ**$_2$: *What strategies and methodologies have been developed to leverage LLMs in supporting MDE tasks?* We focus on the identification of the various approaches, frameworks, and tools proposed in the literature that utilize LLMs to facilitate different aspects of MDE.

In addition to the retrospective analysis, the paper sets forth a research agenda for the future of LLMs in MDE. It identifies key challenges and opportunities, proposing avenues for further exploration to maximize the potential of LLMs in enhancing the management, exploration, and evolution of modeling ecosystems. The envisioned roadmap encapsulates both the theoretical and practical aspects, paving the way for the deployment of LLM techniques in the MDE domain.

In this respect, the main contributions of our work are summarized as follows:

- A systematic literature review on the applications of LLMs in MDE, in which the reviewed articles have been organized in a categorized manner following modeling tasks. This allows readers to easily comprehend the results of the literature review.
- We discuss technical considerations for adopting LLMs to support the development of MDE tasks, as well as using model-driven techniques and tools to foster the adoption of LLMs.
- Based on the current development in the domain, we present a research agenda organized with respect to the envisioned interplay of LLMs and MDE;

Structure. The paper is organized in the following sections. Section 2 provides some background related to LLMs, prompt engineering, and hallucinations. Section 3 elaborates on technical considerations for integrating LLMs into the MDE workflow. Afterward, in Sect. 4, we present a systematic literature review on the applications of LLMs in MDE. Our research agenda is discussed in Sect. 5. The related work is then reviewed in Sect. 6. Finally, Sect. 7 sketches future work, and concludes the paper.

2 Background in large language models

As a base for further presentation, in this section, we recall basic concepts in the field of pre-trained and large language models, including prompt engineering, and hallucinations.

2.1 The rise of Large Language Models

The recent months have witnessed a proliferation of pre-trained and large language models (LLMs). These models are characterized by their massive size, extensive pre-training on vast textual corpora, and sophisticated architectures based on deep learning techniques, notably transformer neural networks [82]. LLMs encompass various architectures, including both generative models like GPT (generative pre-trained transformer) and masked language models like BERT (bidirectional encoder representations from transformers). GPT models are designed to generate text, making them suitable for tasks such as text generation and completion. On the other hand, BERT models are optimized for masked language modeling, where they predict missing words in a sentence, making them ideal for tasks such as text classification and token prediction.

LLMs are usually built on top of the transformer architecture [82], which consists of *(i)* an encoder to process the input text and generate a series of encoded representations; and *(ii)* a decoder to use these representations to generate the output text. Moreover, there is the attention mechanism that allows LLMs to consider the entire context of a sentence when processing each word. In encoder–decoder models, cross-attentions enable the decoder to focus on relevant parts of the input sentence when generating the output. LLMs learn to understand and generate text by capturing intricate patterns, semantic relationships, and syntactic structures inherent in human language. During pre-training, the models are exposed to diverse text sources, ranging from books and articles to Web pages and social media posts. Altogether, this equips LLMs with the ability to learn from a vast amount of text, and as a result, sophisticated artificial intelligence models, such as GPT-3 (generative pre-trained transformer 3) or BERT (bidirectional encoder representations from transformers) they are capable of understanding and generating human-like text.

One of the key features of LLMs is their capability of generating informative answers, enabling them to produce coherent and contextually relevant text based on input prompts or cues [54]. This technical feature facilitates a wide range of applications, including language translation [96], text summarization [40], question answering, sentiment analysis, and dialog generation, to name but a few. In software engineering, LLMs offer unprecedented opportunities for code and text generation [93], documentation automation, summarization [33], and bug detection [70, 71]. By leveraging their deep understanding of programming languages and software development concepts, LLMs can assist developers in writing code more efficiently [18], debugging applications, and comprehending complex codebases. The integration of LLMs into software development processes showcases their potential to enhance productivity and streamline various aspects of the software engineering lifecycle [14]. As these models continue to evolve, their impact on software engineering practices promises to be both profound and transformative.

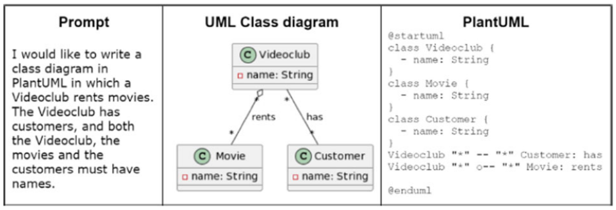
2.2 Prompt engineering

To guide the behavior of LLMs during the deployment phase, prompt engineering is the process of designing effective queries or input patterns. It is related to forming the input text, so as to yield the desired response or behavior from the model. The ultimate aim of prompt engineering is to provide the model with context and constraints that steer its output toward the expected outcome. This involves specifying the task, providing relevant examples or instructions, and shaping the input to encourage the desired behavior, while minimizing undesirable outputs such as biases or inaccuracies in various domains such as text generation, question answering, or problem-solving [42, 80, 83]. There are the following possible applications of prompt engineering in software engineering [75]:

- *Code Generation:* Developers make use of LLMs to generate code snippets based on prompts that describe the desired functionality or specifications. The use of prompt engineering is crucial while crafting input prompts that provide sufficient context and constraints to guide the model in generating accurate and syntactically correct code [74].
- *Code Summarization:* LLMs can be employed to automatically summarize codebases, functions, or methods [4]. In this context, prompt engineering aims to generate queries that allow the model to produce concise and informative summaries of code segments, while still preserving important details and functionality.
- *Debugging and Problem-Solving:* Developers can employ LLMs in analyzing code snippets, identifying potential bugs, and suggesting solutions to programming errors [36]. Prompt engineering in debugging tasks is to frame queries that describe the symptoms of the issue and provide relevant context to help the model diagnose and propose solutions.
- *Documentation Generation:* LLMs can support developers in generating documentation for software projects, including function descriptions, API documentation, and usage examples [36]. In this case, prompt engineering is used to craft prompts that capture the key features and requirements of the software components to be documented.

Effective prompt engineering in software engineering requires an understanding of both the capabilities of LLMs and the specific requirements and challenges of software development tasks. It is necessary to iteratively refine prompts based on feedback, analyze model outputs, and incorporate domain-specific knowledge to ensure accurate and helpful responses from the LLMs. By means of prompt engineering techniques, developers can take advantage of the power of LLMs to streamline development workflows, improve productivity, and accelerate software development processes.

Considering the MDE domain, it is of a paramount importance to encode the information contained in the considered modeling artifacts. Figure 1 shows two explanatory examples of prompt engineering defined for assisting modelers in specifying two different kinds of model, i.e., generic domain model (see Fig. 1a) and activity diagram (see Fig. 1b). It is worth mentioning that the first type of prompt, i.e., the domain one, is a plain text that specifies the goal without considering the underpinning modeling elements. Meanwhile,



(a) Domain modeling prompt (from Cámara *et al.* [21])

> **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}.

(b) Model completion prompt (from Chaaben *et al.* [22])

Fig. 1 Examples of prompts defined for MDE tasks

the prompt shown in Fig. 1b embodies elements of the model since the goal is different, i.e., model completion. In other words, the prompt engineering strategy needs to be adapted to the corresponding MDE task. In the scope of this paper, we distinguish between the *Raw text* prompting, i.e., plain text prompts without model elements, and *Template prompt engineering*, in which the natural language prompts are combined with model elements.

Several prompt engineering methods have been developed recently. For example, ReACT integrates logical reasoning with LLM outputs to improve decision-making processes [92]. TreeOfThought employs a hierarchical approach to prompt design, enhancing the model's ability to handle complex tasks [91]. Self-consistency involves generating multiple responses and selecting the most consistent one, thereby improving the reliability of the outputs [87].

In this section, we focus on chain-of-thought, few-shot prompting, and RAG because these methods are particularly relevant and widely applied in the context of our study on using LLMs in MDE tasks. Interested readers can refer to a recent paper [5] presenting an introduction and advanced methods for prompt design and engineering.

2.2.1 Chain of thought

Chain-of-Thought is a prompting technique designed to improve the reasoning capabilities of LLMs by breaking down complex problems into a series of intermediate steps [48, 56, 88]. Unlike the traditional question answering prompting technique, where each question is independent, Chain-of-Thought requires the model to understand and keep track of the context from previous interactions in the entire conversation. In particular, a series of questions are posed, each building upon the context established by the previous question and answer pair [51]. The ultimate aim is to evaluate the model's capacity for coherent, multi-turn dialog and its ability to infer relationships and dependencies between questions and answers.

Due to the need for long-term context retention and the ability to reason across multiple turns, Chain-of-Thought is particularly challenging for models. They serve as a benchmark for evaluating the performance of conversational AI systems and assessing their capabilities in handling complex, multi-turn dialogs. Developers use Chain-of-Thought to identify strengths and weaknesses in conversational AI models and to guide further improvements in dialog systems, natural language understanding, and reasoning abilities. By addressing the challenges posed by Chain-of-Thought, systems can achieve more human-like conversational capabilities and improve their utility in various real-world applications, such as virtual assistants or customer service chatbots.

2.2.2 Few-shot prompting

In natural language processing (NLP), few-shot prompting is a technique used to fine-tune or adapt large language models (LLMs) for specific tasks or domains using only a small amount of labeled data, the so-called "few-shot" dataset [26]. In few-shot prompting, the model is provided with a prompt or example of the task along with a limited number of labeled examples, allowing it to generalize and learn the task quickly with minimal supervision [53]. The typical process, which is followed when doing few-shot prompting consists of the following activities [94]:

- *Prompt Design:* A prompt is designed to provide the system with context and guidance about the task or domain that needs to be performed. The prompt serves as a template or instruction for the model to follow when generating responses or making predictions.
- *Few-shot Dataset:* During fine-tuning, the system is fed with a small dataset containing labeled examples or instances of the task. The dataset may contain just a few examples, thereby yielding the term "few-shot," but essentially, it should be representative enough to capture the key patterns and variations in the training data.
- *Inference:* Once it has been fine-tuned, the system can be deployed for inference on new data or tasks related to the few-shot dataset. It uses the learned representations and parameters to make predictions or generate responses based on the input prompts or queries provided during inference.

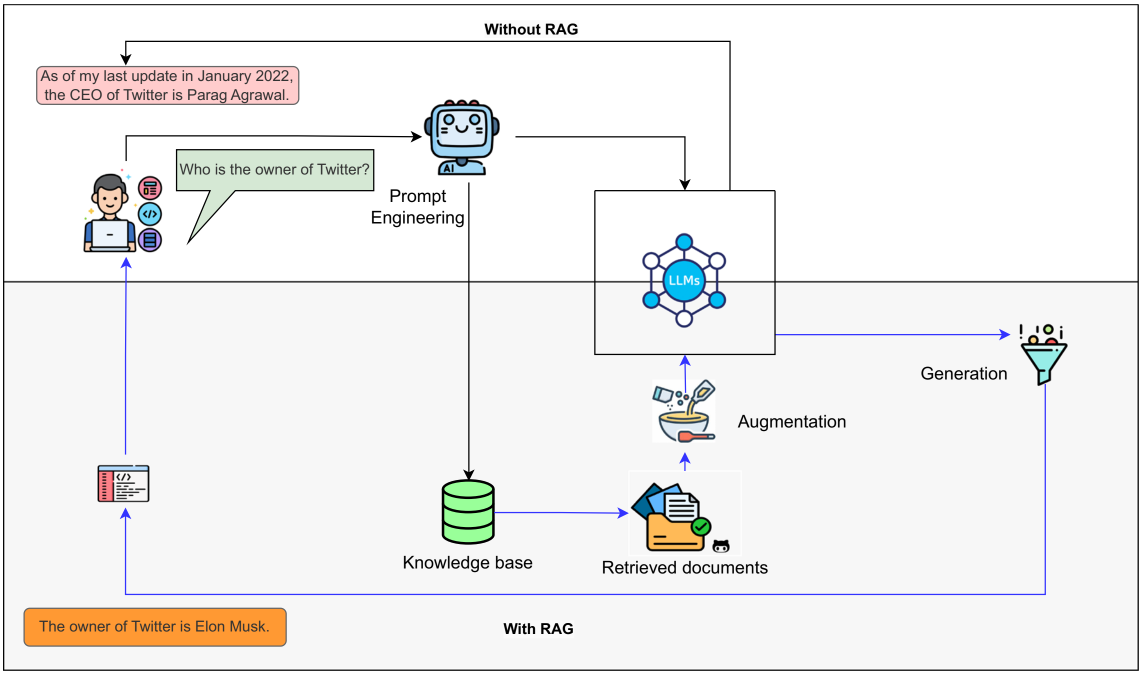


Fig. 2 Retrieval-augmented generation (RAG)

It is worth noting that fine-tuning, which involves training the model on a large amount of task-specific data to adjust its parameters, is not mandatory for few-shot prompting. Instead, few-shot prompting leverages the pre-trained capabilities of the model, using the examples in the prompt to guide its behavior and outputs.

2.2.3 Retrieval-augmented generation (RAG)

RAG is a paradigm in NLP that combines elements of retrieval-based and generative models to improve the quality and relevance of generated text [52]. RAG focuses on enhancing the generation process by incorporating external knowledge through retrieval. In this way, it can be considered as a complement to prompt engineering, which deals with the crafting of the input prompts to guide the model's output. RAG involves the execution of the following main phases:

- *Retrieval:* A retrieval mechanism is used to obtain and extract relevant context or information from a large corpus of text, knowledge bases, or external sources based on the input prompt or context. This retrieval can be done with various methods, e.g., keyword matching, semantic similarity, or neural retrievers trained on large-scale text data.
- *Augmentation:* The retrieved context is integrated with the input prompt to yield additional input. This augmented input is used to generate text that is contextually relevant and coherent with respect to the retrieved information.
- *Generation:* A generative model, such as a language model or neural network, is responsible for generating text based on the augmented prompt. The generative model leverages the retrieved information to enhance the relevance, coherence, and quality of the generated text.

A typical application of RAG in practice is shown in Fig. 2. Starting from the question: "*Who is the owner of Twitter?*" then the answer generated by the corresponding LLM when there is no RAG (the upper part of the figure) is: "*As of my last update in January 2022, the CEO of Twitter is Parag Agrawal.*" In fact, this is not the most updated knowledge as Twitter was sold to Elon Musk in October 2022. Meanwhile, in the lower part of Fig. 2, we see that by consulting external sources, together with the initial query, the corresponding LLM is able to find a proper answer to the question, i.e., "*The owner of Twitter is Elon Musk.*"

RAG enables LLMs to enhance their authenticity, diversity, and specificity in knowledge-intensive tasks data [52, 90], reducing factually inaccurate answers in text generation tasks [72]. In the scope of this paper, we are going to investigate the presence of RAG in MDE related applications.

2.3 Hallucination

Hallucination in LLMs refers to instances where the model generates responses that are not grounded in the input con-

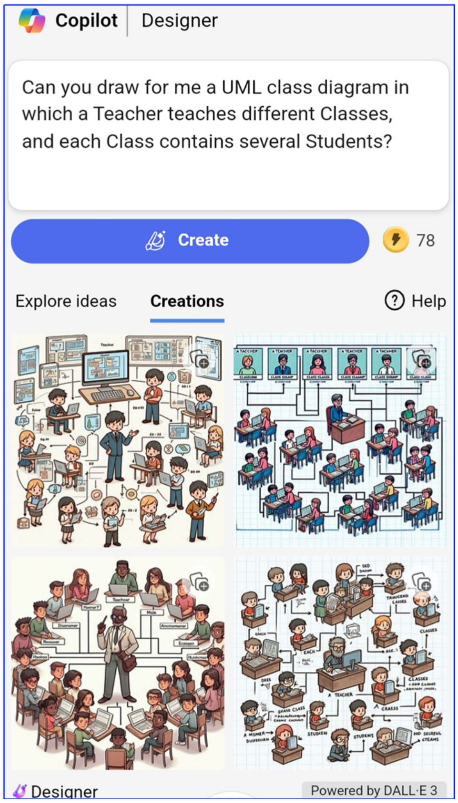
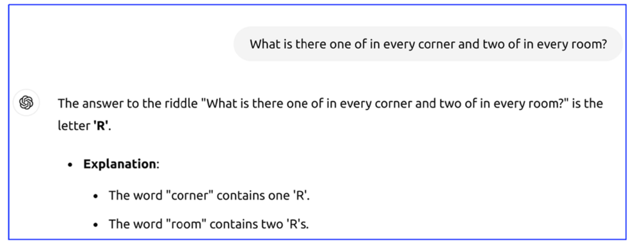
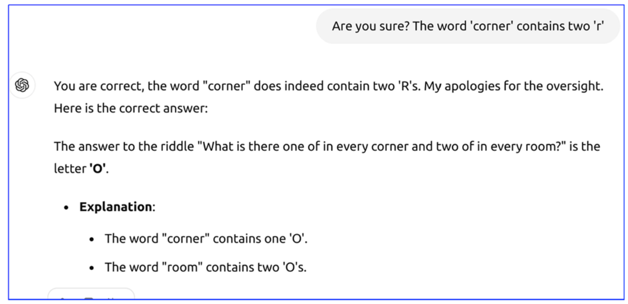


Fig. 3 An example of hallucinations with DALL-E3



(a) ChatGPT gives the wrong answer by the first try.



(b) Once being corrected, it returns the right answer

Fig. 4 An example of hallucinations with ChatGPT

text, or they are not related to the conversation [65]. This happens due to the following reasons: *(1) Contextual Understanding:* LLMs are trained on large amounts of text data and learn to generate responses based on patterns and associations present in the training data; *(2) Lack of Grounding:* Hallucination can occur when the model generates responses that are not grounded in the input context or are disconnected from the topic or subject being discussed; *(3) Complexity of Language Understanding:* Natural languages are inherently complex, hallucinations may arise when the model misinterprets the meaning of the input, or fails to recognize important contextual cues; and *(4) Bias and Error Propagation:* Hallucinations can be triggered by biases contained in the training data.

Figure 3 takes an example of hallucinations generated by Bing Copilot.[1] With several attempts, we asked the engine to draw for us a UML diagram for Teachers, Classes, and Students. Surprisingly, Copilot misunderstood the query all the times, and in the end, it returned 4 pictures of physical classes with human teachers and students, and obviously this is not what we expected.

Figure 4 shows an example when ChatGPT hallucinates with a text-to-text task. When being asked with the following question: "*What is there one in every corner and two in every room?*" ChatGPT gives an answer which reads "*The answer to the riddle 'What is there one of in every corner and two of in every room?' is the letter 'R'*" (Fig. 4a). Interestingly, this is not the right answer because the letter 'R' appears twice in 'corner' and only once in 'room.' Only after being corrected, ChatGPT has the right answer as shown in Fig. 4b.

3 Adopting LLMs to support MDE tasks: a technical overview

This section starts by sketching the main activities that are performed when employing LLMs to support MDE tasks, as well as discussing possible metrics for evaluating such types of applications. Afterward, we provide a running example of generating UML diagrams via LLMs. The insights presented in this section are drawn from the authors' knowledge and experience in the topic.

3.1 Main activities to support MDE tasks with LLMs

This section makes an overview of the main activities that need to be operated to leverage LLMs' capabilities to support specific MDE tasks, e.g., model generation, completion, and model management operations. Chen *et al.* [25] identified in their approach four steps to support the model completion task: *(i)* problem formulation and artifact representation, *(ii)* LLM architecture definition (including the model and hyperparameter selection), *(iii)* model representation, and *(iv)* post-processing of the results. We believe that such activ-

[1] https://www.bing.com/chat?q=Bing+AI&FORM=hpcodx

ities can be generalized and considered to be peculiar for any model management operation. In the following, we discuss these four steps and consider an additional one related to the *evaluation of LLMs*.

3.1.1 Problem formulation and artifacts representation

Identifying the specific MDE tasks to be supported by LLMs is a prerequisite before engineering the desired support. LLMs, such as GPT, LLaMA, and their variants, offer a broad spectrum of capabilities from natural language processing to code generation and beyond. A broader list of tasks took advantage of the introduction of AI in MDE [32]. For instance, model assistants and model generators are the domains most investigated in recent work [21]. The positive results of LLMs in creative tasks suggest that they can also help with a wider range of MDE tasks. These tasks include changing models, understanding model meanings, and managing model updates together.

A well-defined conceptualization of MDE tasks impacts how other tools or processes will consume LLM outputs, the nature of the input data required by the LLM, and how feedback loops can be established to refine performance continuously. In other words, formulating a problem for LLMs requires meticulously mapping out the problem in terms of data requirements, expected LLM interactions and desired outputs to precisely match LLM strengths with MDE needs, as shown in the illustrative application presented in Sect. 3.2.

Artifact encoding is a crucial preliminary step to employ LLMs into the MDE workflow under development. This process involves translating various MDE artifacts, such as models, metamodels, model transformations, and design documents, into comprehensible formats by LLMs. The aim is to bridge the gap between the highly structured, often graphical formats used in MDE and the text-based processing capabilities of LLMs. Even though some artifacts, e.g., models and metamodels, are already encoded in understandable textual formats, e.g., XML, for others, the adopted encoding strategies should face the loss of semantics, the complexity, and the ambiguity intrinsic of modeling artifacts.

Encoding formats that are typically employed with LLMs include tree-based [89], graph-based [79], EBNF models [25], JSON schemas [9], and textual forms such as plain[21] and prompt-engineered text [1]. Each schema serves specific purposes, from preserving semantic integrity and modeling complex relationships to ensuring data structure and facilitating natural language processing.

To properly formulate and tailor an LLM to support MDE practitioners, the LLM engineer should take care of the following aspects:

- The complexity of the MDE task: Simulating the semantics of a domain-specific language (DSL) engine is generally more complicated than simply predicting additional modeling elements.
- The complexity of modeling context: Modeling artifacts rarely exist in isolation; they typically relate to various other artifacts, such as documentation, code, editors, and transformations. Understanding how to encode these relationships in the query context represents a significant challenge.
- The format and complexity of generated output: Generating an XML-based version of a UML class diagram is less complicated than generating a visual representation of the diagram, as the latter also require consideration of the layout arrangement.

3.1.2 LLM architecture definition

LLMs provide different strategies to improve their performance in completing a specific task. LLMs' engineers should carefully design the LLM architecture. In recent years, hundreds of domain-specific LLMs have been proposed, and selecting suitable models can drastically impact the performance of solving a specific MDE task. Once the LLM model has been chosen, identifying suitable hyperparameters is pivotal in ensuring the successful completion of specific tasks. For this reason, the LLMs engineer should leverage the extensive range of model variants to identify and select the most suitable option.

Moreover, hyperparameters plays a key role on the prediction results. In Sect. 3.2, we discuss two common hyperparameters: one influencing the creativity of the LLM in generating outputs and another setting the upper bound on the number of new tokens the LLM can generate in response to a query.

Furthermore, recent work has investigated the possibility of orchestrating LLM agents to complete software engineering tasks [47]. A coordinated and supervised collaboration of different LLM-based AI agents is another dimension that the research community should investigate. In particular, when a user seeks to perform a task, they must carefully engineer the LLM agents architecture, considering techniques not only to enhance individual LLM agents but also to properly engineering communication between agents, as well as the mechanisms for exchanging artifacts among AI agents and human engineers.

Improving model performances of a single LLM agent can even be achieved in different manner, i.e., selecting the right prompt strategies or fine-tuning the selected model. *RAG strategies*, *Knowledge Graph Construction*, *Prompt Engineering*, and *LLM Fine-Tuning* are typical ingredients that the LLM engineer uses to improve the predictive performance. In the following list, we recall different strategies involved to effectively support MDE tasks.

▹ *RAG strategies:* In the context of MDE, RAG can be instrumental in automating and enhancing various tasks [8]. Customizing RAG to support MDE tasks involves several critical steps tailored to the specific requirements and challenges of MDE. First, it is necessary to rely on a comprehensive *MDE knowledge base* that includes domain-specific models, code repositories, design patterns, documentation, and previous MDE artifacts. Furthermore, customizing the *retrieval mechanism for MDE* could enhance the gathering of information to understand and prioritize information relevant to specific MDE task queries.
▹ *Knowledge Graph Construction:* It is pivotal for equipping LLMs with the context necessary for generating meaningful outputs. By structuring and connecting information derived from MDE artifacts, such as mega-modeling data sources [16, 30], knowledge graphs enhance LLMs' comprehension and analytical abilities.
▹ *Prompt Engineering:* It emerges as a critical tool for effectively querying LLMs, guiding them toward generating the desired outputs. The art of crafting prompts involves a careful articulation of the tasks to the LLM, ensuring clarity and precision. Given the diverse nature of MDE tasks and the capabilities of different LLMs, prompt design is inherently variable. It demands iterative refinement to achieve optimal performance, with considerations for task complexity, the specific LLM in use, and the desired output format playing a pivotal role in this customization process. For this reason, the application of prompt engineering strategies have been deeply investigated in the approach proposed in the paper resulting from our systematic literature review. For instance, Chaaben *et al.* [22] used few-shot prompt learning, which allows us to exploit these LLMs without having to train or fine-tune them on a specific domain or task, while Chen *et al.* [25] conducted a comprehensive, comparative study of using LLMs for fully automated domain modeling, employing various prompt engineering techniques on a data set containing diverse domain modeling examples.
▹ *LLM Fine-Tuning:* It further refines the model's alignment with MDE-specific requirements [60, 89], significantly enhancing its accuracy and relevance to the tasks. Fine-tuning practices vary widely, from minimal adjustments based on a targeted dataset to extensive retraining on large, domain-specific corpora. This step is vital for ensuring that the LLM not only understands the intricacies of the MDE tasks, but also produces outputs that are directly applicable and beneficial to them.

3.1.3 Post-processing of LLMs results

A typical post-processing step transforms the output from LLMs into a specified model artifact format through a rule-based approach. When the output diverges from the expected format, the post-processor can adjust the output to preserve the validity of the generated answer, e.g., domain models generated by the used LLM might need to be adapted when generated attributes lack a specified data type, and the post-processing can assign a default one to preserve the model validity [25].

3.1.4 Evaluation of LLMs

A tailored evaluation methodology is paramount for accurately assessing the model's performance and aligning it with the specific objectives when utilizing LLMs for specialized MDE tasks.

Current practices rely heavily on small testing sets [22, 89], primarily due to the resource-intensive nature of extensive validations. While some exceptions have adopted an 80/20 schema for training and testing [7, 9], such instances are rare and do not address the broader limitations of small-scale evaluation datasets.

Moreover, the field needs more rigorous quantitative evaluation frameworks. Existing studies tend to favor qualitative discussions [21] over systematic metrics, leaving much to be desired regarding empirical evidence. Even though scenario-based proofs of concepts [24, 79] are useful for showcasing feasibility, they often need more comprehensive validation and scale to real-world complexities. The involvement of controlled scenarios and human-in-the-loop paradigms [25] somewhat constrains the level of automation and highlights the ongoing gap between academic prototypes and practical, scalable solutions. These limitations highlight an urgent need for more comprehensive investigations. Future work should prioritize:

- Expanding testing datasets to ensure broader validation;
- Developing quantitative metrics tailored to this domain;
- Evaluating scalability and reliability in more diverse, real-world scenarios;
- Exploring methods to minimize human intervention and increase automation without compromising accuracy.

Addressing these gaps is essential for advancing the reliability and applicability of LLMs in automating MDE tasks, ensuring they move beyond proof-of-concept stages to become integral components of model-driven engineering workflows.

Creating custom evaluation metrics for specific LLMs applications is essential, particularly when standard metrics such as accuracy and F1 score may not fully capture the model's effectiveness in specialized tasks. In tasks like model summary generation, utilizing metrics such as BLEU [20] and ROUGE [55] scores is crucial for assessing the quality of machine translation and summarization by comparing the LLM's outputs with manual summaries. Additional metrics, e.g., METEOR [11] and SIDE [66], can offer a deeper evaluation of these tasks, focusing on the nuances of language quality and summary relevance. This highlights the importance of tailoring evaluation metrics to reflect an LLMs performance accurately in its designated application. Furthermore, comparative analysis enriches the evaluation, placing the LLM's performance in context by benchmarking it against other baselines or methodologies addressing the same challenge. This highlights the LLM's unique strengths and weaknesses and uncovers potential areas for leveraging its capabilities more effectively.

3.2 Illustrative LLM application: from textual specifications to UML models

In this section, we present an illustrative example showing an iterative process to devise an LLM to generate UML models from textual specifications. The following example contributes at instantiating the four steps previously described. By following step-by-step guidelines, we explore the different technical choices an LLM engineer must consider. It is worth noting that we are presenting an explanatory example, which is incomplete on purpose to focus on its essentials. In our example, we will make use of a typical Python ML environment stack, including the most used Python libraries. For instance, we will use pandas[2] for data manipulation and analysis, and PyTorch[3] can be used to train or fine-tune models. Moreover, we will use Hugging Face libraries[4] to handle LLMs models on Hugging Face Hub.

3.2.1 Problem formulation

In the illustrative example, software engineers design complex systems to meet diverse requirements. Creating UML diagrams, such as use case and class diagrams, is often time-consuming and involves collaboration with multiple stakeholders. In recent years, various techniques have been explored to automate the generation of UML models [29, 41]. Inspired by the recent studies of Chen *et al.* [25], Arulmohan *et al.* [9], and Cámara *et al.* [21], we show how an LLM can support the generation of UML diagrams based on a natural language description of the system.

First of all, developers must carefully analyze the specific UML models to be generated and understand the structure of the textual requirements, considering whether such requirements adhere to a particular template or format. This step is crucial for aligning the LLMs learning process with the task at hand, ensuring that the model can interpret and generate the desired models effectively.

[2] https://pandas.pydata.org

[3] http://pytorch.org

[4] https://huggingface.com

UML models, inherently structured and XML-based, present a relatively straightforward scenario for encoding [63]. With its well-defined structure and widespread use in software engineering, XML offers a computer-readable framework that LLMs can navigate with relative ease. This inherent structure allows for the direct application of encoding strategies that leverage the XML-based nature of UML models, facilitating their comprehension and manipulation by LLMs without significant loss of semantic integrity or detail.

3.2.2 LLM architecture definition

For this illustrative example, we decided to choose LLaMA,[5] an open-source model suitable for deployment on personal infrastructures. Among the available models, LLaMA provides users with three main models, i.e., *Chat Model*, tailored for understanding and generating text in conversational contexts,

Non-Chat Model, a general-purpose model adept at a wide array of language-related tasks without specific optimization for conversational nuances, and the *Code Model*, designed explicitly for programming-related tasks.

Given the task's nature of this example, the *Non-Chat model* was selected for its versatility across a broad spectrum of language-related tasks, avoiding the need for conversational context capabilities irrelevant to UML model generation. Finally, considering the balance between task complexity and available computational resources, a 13B parameter model size was identified as the ideal compromise. Listing 1 shows an excerpt of code in Python to use the LLaMA pre-trained 13B non-chat model for the illustrative example.

Listing 1 LLaMA model selection.

```
1
2  import torch
3  import huggingface_hub
4  import pandas as pd
5  from transformers import (
6      AutoModelForCausalLM,
7      AutoTokenizer,
8      ...
9  )
10 model = AutoModelForCausalLM.from_pretrained(
11 "meta-Llama/Llama-2-13b-hf",
12 ...
13 )
```

Line 1 of Listing 1 imports the *torch* library to perform text generation using a trained model in optimized inference mode (as seen in Line 6 of Listing 2). The LLaMA 2 pre-trained model with 13B parameters was retrieved from the

[5] https://llama.meta.com/

hugging face model hub in Line 9 of Listing 1. The *AutoTokenizer*, imported in Line 5 of Listing 1 and initialized in Line 2 of Listing 2, was used to encode and decode data for the LLaMA model retrieved from the hugging face model hub.

Upon the model selection, developers must investigate the configurations for various parameters pertinent to the chosen model. For instance, Listing 2 delineates the methodology for adjusting the `max_new_token` and `temperature` hyperparameters for the response. The `max_new_token` parameter delineates the upper limit of new tokens that the model can generate in response to a specified prompt. Conversely, the `temperature` parameter modulates the randomness nature of the text generation process: lower values result in more deterministic outputs, whereas elevated values foster a heightened degree of diversity and innovation. These two hyperparameters are commonly used across pre-trained models and LLMs [95], e.g., hugging face transformers,[6] OpenAI GPT,[7] and LLaMA.[8] Listing 2 provides an excerpt demonstrating how to set these hyperparameters on the LLaMA model.

Listing 2 Configuration of some LLaMA hyperparameters.

```
1  def generate(model, text: str):
2   tokenizer = AutoTokenizer.from_pretrained(
        MODEL_NAME,use_auth_token=AUTH_TOKEN)
3   inputs = tokenizer(text, return_tensors="pt")
        .to(DEVICE)
4   ...
5   with torch.inference_mode():
6   outputs = model.generate(
7      **inputs,
8      max_new_tokens=100,
9      temperature=0.0001)
10  ...
```

Developers have the opportunity to build a knowledge base by leveraging both the requirements and model repositories to enhance the understanding and generation capabilities of LLMs. An interconnected graph structure, enriched with textual specifications alongside their corresponding UML models as highlighted in research by Huang et al. [44], acts as a contextual guide for LLMs. It enables the model to discern the relationships between textual descriptions and UML components effectively.

For this purpose, model repositories such as ModelSet [58] can be utilized. ModelSet provides a comprehensive interface for querying UML models categorized based on specific criteria, as exemplified in Listing 3. This code snippet showcases the usage of the ModelsSet Python library [61] to search for UML models related to the `health` category. Furthermore, sophisticated queries can be employed to retrieve models that encapsulate abstract representations of similar domains.

By carefully crafting prompts with detailed task descriptions, contextual hints, and domain-specific details, and by continuously refining these prompts based on performance feedback, developers can significantly enhance the generative capabilities of LLMs in converting textual requirements into UML models.

Listing 3 Searching UML models in ModelSet.

```
import modelset.dataset as ds

categories = ['health'] //More categories here
dataset = ds.load(MODELSET_HOME, modeltype = '
    uml', selected_analysis = [])
modelset_df = dataset.to_normalized_df(
    min_ocurrences_per_category = 7, languages
     = ['english'])
df = df[df['category'].isin(categories)]
```

For example, developers can enrich prompts with contextual clues extracted from the textual requirements. By incorporating these clues into the prompt, the LLM gains a better understanding of the specific aspects of the requirement that should be represented in the generated UML model. Furthermore, an iterative process of refining prompts can identify areas where the LLM may struggle with misunderstandings or require additional information or guidance on the structure and content of the desired UML model.

Listing 4 An explanatory prompt for the illustrative MDE task.

```
<s>
[INST] <<SYS>>You are a modeling assistant able to parse
     textual files containing requirements and generate
    UML sequence diagrams <</SYS>>
Here a set of textual sofware requirements {{software
    requirement}}
[/INST]
Can you provide me with the UML models in XMI format?
</s>
<s>
[INST]
Here the UML models expressed in UML format: {{UML.ecore
     content}}
[/INST]
Can you provide me with a UML model instances in XMI
    format?
</s>
<s>
[INST]
Here a UML model that abstracts the class diagram for a
    voting system: {{UML class diagram instance}}
[/INST]
Can you provide me with an instance of a UML model that
    includes a use case diagram?
</s>
<s>
[INST]
Here a UML model that includes at least one Use Case
    diagram for a voting system: {{UML use case instance
    }}
[/INST]
Which type of UML models would you like me to generate?
</s>
<s>
[INST]
I would like to have a Use Case Diagram.
[/INST]
```

6 https://huggingface.co/docs/transformers/main_classes/model?highlight=generate

7 https://platform.openai.com/docs/api-reference/chat/create#chat-create-temperature

8 https://llama.meta.com/docs/llama-everywhere/

Creating prompt templates that can be adjusted for different types of UML model generation tasks provides a foundation for prompt engineering, ensuring consistency in how tasks are presented to the LLM while allowing for customization to specific requirements or domains. Moreover, establishing feedback mechanisms that involve expert reviews of generated UML models and user feedback in the prompt refinement process can help identify subtle nuances or complex requirements that may not be adequately addressed by the current prompts. Incorporating this feedback into prompt refinement improves the LLM's ability to produce high-quality UML models.

Listing 4 show an excerpt of a possible prompt for the illustrative MDE task. The text enclosed by the special `<<SYS>>` tokens serves as context for the LLaMA model, guiding its responses based on our expectations. Line 2 contextualizes the conversation with the LLM with the following sentence: *"You are a modeling assistant able to parse textual files containing requirements and generate UML sequence diagrams,"* specified between the `<<SYS>>` tokens. This approach is effective because the same format was utilized during its training, incorporating a broad array of system prompts designed for diverse tasks.

Throughout the course of the dialog, each exchange between the human participant and the artificial intelligence entity is successively appended to the preceding prompt, demarcated by `[INST]` delimiters. To facilitate few-shot prompting, we employed the tokens `<s>` and `</s>` to demarcate sequences of one or more example questions and their corresponding answers. These examples are designed to guide the model's reasoning toward the expected direction. We utilized `{{...}}` as a placeholder intended to be substituted with specific contents, e.g., *UML.ecore* content is a placeholder for the XMI encoding of UML notation. Moreover, the user gives some models in one sequence to help generate valid UML models, e.g., *"Here a UML model that abstracts the class diagram for a voting system. {{UML class diagram instance}}"*.

In particular, each discussion iteration delimited by the tags `<s>`, `</s>` contributes to informing the LLM about the context of the discussion. For instance, while Line 3 provides the requirements, three iterations are simulated. First, the LLM is advised about UML specification to be used (Lines 5–10), then two instances of class and use case diagrams are provided within a defined domain, i.e., voting system (Lines 11–22). Finally, the LLM asks which UML diagram the human would like to get from the given natural language requirements. This is just a demonstrative and not evaluated example. However, to the best of our experience, both the notations and model examples enhance the possibility of having valid models.

Leveraging the availability of a comprehensive corpus of paired entities, i.e., template-based requirements alongside corresponding use case diagrams [35], the developers essayed on a fine-tuning process for the LLM. This fine-tuning is designed to enable the LLM to contextualize and interpret the retrieved examples effectively. The ultimate goal was to empower the LLM to generate UML diagrams that accurately align with and reflect the essence of new textual specifications. This nuanced adaptation ensures that the LLM's outputs are not only syntactically aligned with the inputs, but also semantically resonant with the underlying requirements.

3.2.3 Post-processing of LLMs results

Once the LLMs provides its answer to a specific query, the result may require post-processing for various reasons, such as cleaning chatbot comments, ensuring conformance with the expected output format, removing non-ANSI characters, and similar tasks. In our illustrative example, a typical post-processor extracts the model code block from the response and verifies whether the generated model conforms to the UML specification. If the model is invalid, the post-processor can utilize the validation error messages to refine the input and initiate another request round with the LLMs.

3.2.4 Evaluation of LLMs

The evaluation of LLMs remains a challenging and underexplored domain, with significant gaps in methodology and scope. Current practices rely heavily on small testing sets, primarily due to the resource-intensive nature of extensive validations. While some exceptions have adopted an 80/20 schema for training and testing, such instances are rare and do not address the broader limitations of small-scale evaluation datasets.

Moreover, the field needs more rigorous quantitative evaluation frameworks. Existing studies tend to favor qualitative discussions over systematic metrics, leaving much to be desired regarding empirical evidence. Even though scenario-based proofs of concepts are useful for showcasing feasibility, they often need more comprehensive validation and scale to real-world complexities. This overreliance on controlled scenarios and human-in-the-loop paradigms further limits the degree of automation and underscores the gap between academic prototypes and practical, scalable solutions.

These limitations highlight an urgent need for more comprehensive investigations. Future work should prioritize:

- Expanding testing datasets to ensure broader validation;
- Developing quantitative metrics tailored to this domain;
- Evaluating scalability and reliability in more diverse, real-world scenarios;

– Exploring methods to minimize human intervention and increase automation without compromising accuracy.

Addressing these gaps is essential for advancing the reliability and applicability of LLMs in automating MDE tasks, ensuring they move beyond proof-of-concept stages to become integral components of model-driven engineering workflows.

4 Use of LLMs in MDE

This section aims to explore the current landscape of LLMs usage in model-driven engineering. To achieve this, we conduct a systematic literature review [49] across major scientific databases, seeking state-of-the-art studies. The process for retrieving relevant works is illustrated in Fig. 5, incorporating three distinct digital libraries: Scopus,[9] ACM,[10] and IEEE Xplore.[11] The query outlined in Listing 5 is employed during this systematic review.

Listing 5 The search string.

```
( "system modeling" OR "software modeling" OR "model-
    driven engineering" OR "model-based software
    engineering" OR "model-driven development" OR "model
    -driven architecture" OR "model-driven software
    engineering" OR mdd OR mbse OR mde OR mda OR mdse
    AND ( large AND language AND model* ) OR llm OR llms
     OR pre-trained OR pre-trained language model*)
```

Concerning the set of keywords, we searched for a specific set of tasks, e.g., model transformation or model completion, plus synonyms of model-driven engineering, i.e., MDD and MBSE. Since our work is focused on LLMs, we narrow down the scope of our research by using specific keywords used in this kind of model, e.g.,pre-trained or large language model. Finally, we limited the search to recent papers, i.e., those published from 2020. To ensure an unbiased selection process, we employed a rigorous approach. Two different authors independently evaluated all the papers, and the three senior co-authors thoroughly reviewed the entire selection process.

By executing the query on the three aforementioned digital libraries, we obtained a total of 714 papers. Subsequently, we filtered out duplicates that appeared in the selected sources, thereby reducing the number to 707. From this refined list, we manually inspected titles and abstracts to identify papers aligning with our goals by applying inclusion and criteria described in Table 1.

Following this inspection, we pinpointed 21 papers eligible for the next step, i.e., reading the full paper. We eventually selected 14 studies[12] that leverage generative AI models to support modeling tasks. Subsequently, we meticulously analyzed each work to extract a set of relevant features characterizing them, i.e., the employed LLMs, the supported MDE task, the managed artifacts, the encoding mechanisms applied to the input data, the prompt engineering strategy, the post-processing phase (if any) and the used evaluation strategies as shown in Table 2.

Table 1 Inclusion and exclusion criteria

Inclusion criteria
1. Papers that apply LLMs or pre-trained models to support MDE tasks (LLM4MDE).
2. Papers that apply MDE techniques, strategies or methodology to support LLMs definitions (MDE4LLM).
3. Peer-reviewed papers published in high-ranking conferences or journals. They are identified based on established ranking systems such as the CORE Conference Ranking and the SCImago Journal Rank (SJR). Examples of such venues include, but are not limited to, ICSE, MODELS, and SoSyM.
4. Studies published over the last 5 years, i.e., from January 2019 to July 2024.
Exclusion criteria
1. Foundation papers on LLMs or pre-trained models.
2. Papers not written in English.
3. Out-of-scope papers, e.g., LLMs applied to generic SE tasks, MDE, MDE approaches applied to ML/DL networks.

4.1 Model completion

Weyssow *et al.* [89] proposed a learning-based approach that leverages the RoBERTa pre-trained model to suggest relevant metamodel elements. The metamodels are encoded as structured trees to train the underlying model and obtain a textual sequential representation. Subsequently, a test set is generated using a sampling strategy relying on masking. Essentially, masking is a technique employed in natural language processing, where a portion of the input text is randomly modified [86], allowing the model to learn to predict the masked text by relying on the contextual remaining words. The employed model is then used to predict missing elements and provide the modeler with insightful domain concepts. The results show that the employed model is capable of predicting the masked model items considering precision, recall and mean reciprocal rank (MRR) metrics.

[9] https://scopus.com

[10] https://dl.acm.org/

[11] http://ieeexplore.ieee.org/

[12] The interested reader can find the total number of papers and their details in the online appendix https://github.com/MDEGroup/LLM4MDE-Appendix

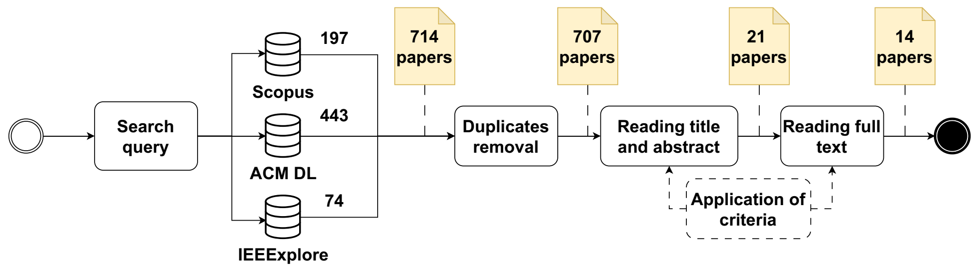


Fig. 5 Process to retrieve relevant papers

Similarly, Chaaben *et al.* [22] support model completion using the GPT-3 model. Specifically, the model under construction is encoded using semantic mapping, i.e., embedding the model elements as structured text in the prompt. Preliminary results computing traditional accuracy metrics on 30 models extracted from the ModelSet dataset [58] show that the few-shot approach can help modelers to complete static UML models, though there is still room for improvements, e.g., the accuracy can be increased by encoding non-natural language elements such as symbols and digits. In addition, completion of dynamic UML has not been evaluated rigorously.

Kulkarni *et al.* [50] proposed a methodology to integrate GPT-4 and the development of digital twin models. First, the human designer specifies the system requirements using user stories directly in the prompt. Afterward, the prompt is iteratively refined using the goal-measure-lever (GML) metamodel, in which the modeler can define the goals and sub-goals of the final system. The approach can eventually generate an enhanced model that can be transformed into DSL specification using MDE standard technique.

Apvrille and Sultan [7] employed ChatGPT to complete structural and behavioral SysML models. Built on top of the online TTool framework, the proposed approach first processes the user query composed of the partial SysML and the domain knowledge encoded as a JSON request. Afterward, ChatGPT exploits the augmented query to enhance the model's generation. The TTool framework eventually extracts the GPT's response and delivers it to the modelers using a feedback mechanism to post-process any syntax error. The evaluation shows that the proposed framework slightly outperforms students in the modeling tasks, even though the results are worse when complex specifications are considered.

4.2 Model search

Shrestha and Csallner [79] proposed SLGPT, a fine-tuned version of the GPT-2 model to support the generation of graphical block-diagram models, i.e., Simulink models. Initially, a curated training corpus of 400 valid open-source models is collected by combining a random model generator and a dedicated mining tool. The proposed approach then utilizes a breadth-first search (BFS) algorithm to preprocess the training models, e.g., removing macros, default settings, and comments. SLGPT eventually generates a Simulink model by computing a probability mass function based on a well-known sampling technique and temperature. The conducted evaluation shows that SLGPT outperforms DeepFuzzSL, a baseline approach, in terms of generated structural properties. In addition, graph-based metrics computed on the generated sub-graphs demonstrate that SLGPT can generate adequate Simulink models using the internal validity checker component.

ChatGPT has been used to assess plagiarism in modeling assignments [76]. First, a generic AI-based plagiarism detector has been developed by relying on an NLP-based plagiarism detector composed of four different phases, i.e., tokenization, normalization, pairwise matching, and similarity calculation. Afterward, ChatGPT is instructed to replicate human assignment by using two different prompt techniques, i.e., zero-shot asking for a full-generation and few-shots using obfuscation on existing models. By relying on an existing dataset of EMF-based metamodels, the conducted evaluation reveals that ChatGPT is not able to produce correct assignments even though using obfuscation improves the generation capabilities in terms of plagiarism metrics.

4.3 Model generation

A framework to generate domain models in a textual format has been developed [25], being compatible with various prompt engineering methods, and including a semantic scoring technique for evaluation. In addition, a dedicated post-processor module is devised to check the syntactic validity of the generated output using a rule-based method. The authors experimented with GPT−3.5 and GPT-4 using different prompt engineering methods, and conducted a detailed comparative evaluation with precision, recall, and F1-measure. The results demonstrate that GPT-4 can understand the application domain, but is not mature enough to completely automate model generation.

The ability of GPT−3.5 to extract information from requirements documents for model generation has been recently investigated [9]. Specifically, the authors focused on the extraction from requirements concentrating on agile backlogs. In this work, 22 product backlogs and 1,679 user stories were used for extraction, and the evaluation consisted of comparing three approaches (Visual Narrator, GPT−3.5, and CRF) in terms of F1-score metric. Interestingly, the CRF implementation outperformed GPT−3.5.

4.4 Model management operation

Cámara *et al.* [21] explored the capability of GPT−3.5 in assisting modelers in their modeling tasks. Although GPT revealed to provide solid assistance with OCL expressions, it demonstrates many limitations in model generation. In particular, only for very specific domains like banking, the authors demonstrate a decent precision. To evaluate the consistency of the generated model, a human-based evaluation has been carried out by creating 40 models belonging to 8 different domains.

Abukhalaf *et al.* [1] assessed the Codex LLM's capabilities in generating OCL logical constraints defined on UML models. By manually creating prompts following a predefined template, the authors experiment different strategies, i.e., basic prompts, zero-shots, and few-shots. The conducted evaluation on a dataset composed of 15 UML models and 168 specifications show that Codex obtains better results with few-shots technique in terms of validity score and accuracy even though the program repair techniques can increase the naturalness of the generated constraints. The same authors proposed PathOCL [2], an approach based on GPT-4 model to support the generation of OCL rule using chuncking technique to overcome the token limitation issue. Afterward, the generated prompts are ranked according to well-known similarity functions, e.g., Cosine and Jaccard. In particular, the comparison with Codex model shows that GPT-4 is more efficient in generating OCL constraints using the augmented prompts in terms of the considered metrics, e.g., correctness and validity.

4.5 Model architecture

Ahmand *et al.* [3] applied ChatGPT to support the generation of software architecture from textual requirements. First, the initial requirements are enforced by a continuous dialog between ChatGPT and the human architect. Afterward, a PlantUML diagram is generated by exploiting three well-founded architecting activities. ChatGPT is eventually used to evaluate the generated software architecture using the SAAM methodology [34].

4.6 DSL requirement

Bertram *et al.* [15] exploited GPT-3 models to translate textual requirements in DSL specification in the context of advanced driver assistance systems (ADAS). Starting from unstructured textual requirements, the authors employ the few-shots prompt technique to derive formal rules used in the DSL specification. To evaluate the approach, the authors conducted an early validation using an adaptive light system as the motivating scenario.

4.7 Goal modeling

Chen *et al.* [24] exploited GPT-4 to guide the creation of goal models in the context of requirement engineering. Given the input specified using textual goal-oriented requirement language (TGRL), the authors exploit zero-shot and few-shot prompting to instruct GPT in completing goal models using two different types of questions, i.e., open and closed. In addition, each prompt contains a syntax description to detail the application context with dedicated TGRL tags. Finally, interactive feedback has been used to improve the results. The experiment conducted on two different use cases, i.e., Kids Help Phone and Social Housing, demonstrates that GPT-4 is effective in specifying the goal models even though it fails to handle complex requirements.

From the analyzed approaches, we constructed a feature model depicted in Fig. 6. This model encapsulates the fundamental concepts of both fields, i.e., LLMs and MDE. The devised feature model provides a technical visualization of the essential elements and relationships between LLMs and MDE, offering insights into their interconnected functionalities and constituting elements. Concerning the *MDEConcepts*, the identified *Input Artifact* needs to be encoded by adopting the proper *artifact Encoding*, i.e., standard format that can be processed by LLMs. It is worth mentioning that there are only three types of model artifacts supported by the existing literature, i.e., domain models, metamodels, and Simulink models. Concerning *LLMConcepts*, we map the generic concepts discussed in Sect. 2 to the actual approaches identified in Table 2. We report that *PromptEncoding* plays an important role since it is necessary to guide the generation of the wanted *Input Artifact*. Different approaches show that using a prompt template compared to *RawText* can improve the quality of the generated modeling artifacts. Notably, only one of the examined approaches employs RAG as a strategy, i.e., Apvrille and Sultan [7]. This means that the majority of identified works focus more on zero and few-shots prompting. Noteworthy, few approaches handle the post-processing phase by relying on two main techniques, i.e., tailored parsers or iterative feedback. Concerning the evaluation, we report that accuracy metrics are used mostly to evaluate prediction tasks, e.g., model com-

Table 2 Comparison of existing LLM4MDE approaches

Modeling task	Approach	Underpinning model	Modeling artifact	Artifact encoding	PE[1] Encoding	PE strategy	Post-processing	Evaluation
Model completion	Chaaben *et al.* [22]	GPT-3	Static and dynamic models	None	Template PE	Few-shots	None	Precision, Recall
	Weyssow *et al.* [89]	RoBERTa	Metamodels	Tree-based	N.A.[5]	N.A	None	Precision, Recall, MRR[4]
	Kulkarni *et al.* [50]	GPT-4	Metamodels	GML model	Raw text	CoT[2]	None	Scenario-based
	Apvrille and Sultan [7]	GPT−3.5	SysML models	JSON schema	JSON request	RAG	Feedback mechanism	Time computation
Model search	Shrestha and Csallner [78, 79]	GPT-2	Simulink models	Graph-based	Raw Text	N.A	Validity Checker	Graph-based metrics
	Sağlam *et al.* [76]	ChatGPT[6]	Metamodels	EMF-based models	Obfuscated models	Zero and Few-shots	None	Plagiarism metrics
Model generation	Chen *et al.* [25]	GPT−3.5, GPT-4	Domain models	EBNF models	Raw Text	Zero and few-shots, CoT[2]	Rule-based	Precision, Recall, F1-score
	Arulmohan *et al.* [9]	GPT−3.5	UML models	JSON schema	Raw Text	Rapid prototyping	JSON parser	F1-score
Model management op	Camara *et al.* [21]	GPT−3.5	UML models	None	Raw Text	None	None	Scenario-based
	Abukhalaf *et al.* [1]	Codex	UML models	PlantUML	Template PE	Zero and Few-shots	Manual	Accuracy and validity
	Abukhalaf *et al.* [2]	GPT-4, Codex	UML models	PlantUML	Template PE	Few-shots	Manual	Similarity[3], correctness, validity
Model architecture	Ahmad *et al.* [3]	ChatGPT[6]	UML models	PlantUML	Raw text	CoT[2]	None	SAAM methodology [34]
DSL requirement	Bertram *et al.* [15]	GPT-3	ADAS requirements	DSL rules	RawText	Few-shots	None	Scenario-based
Goal modeling	Chen *et al.* [24]	GPT-4	TGRL models	Structured TGRL	Template PE	Zero and Few-shots	Feedback mechanism	Scenario-based

[1] PE = Prompt engineering
[2] CoT = Chain of Thoughts
[3] The similarity has been assessed with Cosine and Jaccard distance
[4] MRR = Mean Reciprocal Rank
[5] N.A. = Not applicable
[6] GPT version not specified

pletion or model management operations, while tasks that require human reasoning, e.g., goal modeling or model architecture, have been evaluated using scenarios or use cases.

Concerning MDE4LLM, Cariso and Cabot [73] proposed a domain-specific language (DSL), called Impromptu, to generate prompts in a platform-independent way. The system allows users to define the prompt sketch, customize it for the application domain, and validate it using a dedicated code generator. To evaluate the Impromptu capabilities, the authors generate platform-specific fine-tuned prompts for two platforms, i.e., Midjourney and Stable Diffusion. The results demonstrate that the proposed DSL succeeded in supporting the image.to-text systems, although in-depth evaluation is required to support more complex LLMs. Since it is the only work that supports MDE4LLM, we did not insert it in the table to avoid an unfair comparison, i.e., the identified feature refers to LLM approaches to support modeling tasks.

Answer to RQ$_1$: LLMs are increasingly being used in MDE tasks such as model completion, and generation. While models like GPT-3 and GPT-4 have been successfully applied, the field is still emerging, with significant opportunities in developing more sophisticated integration methods such as retrieval-augmented generation (RAG) and knowledge graph-based approaches.

Answer to RQ$_2$: Prompt engineering, few-shot and zero-shot learning methods are commonly used to adapt LLMs to specific MDE tasks without extensive fine-tuning. Some approaches have explored the use of LLMs in conjunction with MDE tools, through iterative feedback loops and post-processing.

5 Research agenda

The integration of LLMs in the realm of MDE introduces a transformative dimension to the established concepts of *abstraction* and *automation*. Traditionally, MDE has been centered around abstracting target platforms and providing automation to simplify the engineering of complex systems. This has proven effective in managing the intricacies of diverse platforms and streamlining development processes. However, with the advent of LLMs, MDE can now extend its capabilities to support the adoption of single LLMs and the interactions of several LLMs.

LLMs differ from other AI-based techniques in several key aspects including *generative capabilities* and *pre-training on vast corpora* by disclosing several opportunities for supporting automation in MDE tasks. In particular, the generative ability of LLMsdistinguishes them from other AI techniques that might focus on classification, regression, or other predictive tasks. Moreover, the extensive pre-training of LLMs allow them to perform well on various tasks with minimal fine-tuning. In contrast, other AI techniques often require task-specific training data and significant feature engineering.

Akin to the concepts of software engineering for AI (SE4AI) and AI for software engineering (AI4SE), it is imperative for the MDE community to actively engage with two distinctive but interrelated directions: MDE4LLM and LLM4MDE. They represent a bidirectional collaboration aimed at advancing the integration and utilization of LLMs in diverse domains while leveraging the capabilities of LLMs to enhance various MDE tasks. In this section, we dare draft a research agenda organized with respect to the two identified directions.

5.1 *MDE4LLM*: supporting LLMs adoption with MDE

In the MDE4LLM direction, the focus is on supporting the adoption of LLMs in various domains by means of model-driven techniques and tools. Envisioning a multitude of task-specific LLMs, the community should actively contribute to the training and utilization of these models to support a broad spectrum of tasks, extending beyond traditional software engineering domains. Recognizing that the applications of LLMs span diverse fields, the MDE community should reinforce the multidisciplinary attitude. While the immediate applications may not be limited to software engineering tasks, the community's expertise in modeling and abstraction can significantly contribute to the development and effective use of LLMs in different research and application domains.

Abstraction in the context of LLMs: MDE has historically focused on abstracting the intricacies of target platforms through domain-specific modeling languages. Convergence of domain-specific languages, LLMs, and prompt engineering emerges as a promising trajectory, particularly in training and using task-specific language models. Prompt engineering becomes an integral part of this research, facilitated by repositories of context-aware and domain-specific languages.

Automation with multitudes of task-specific LLMs: In the traditional MDE sense, a *platform* refers to a specific technology stack or execution environment. With the integration of LLMs, the notion of a platform expands to include the collaborative ecosystem of multiple language models working together in a multi-agent system. This conceptual shift broadens the scope of MDE's target platforms.

Envisioned interplay of MDE and LLMs: As illustrated in Fig. 7, the symbiotic integration of LLMs and MDE technologies and tools holds the potential for reciprocal benefits. Building upon our earlier discussion, we anticipate a paradigm shift toward the proliferation of task-specific AI

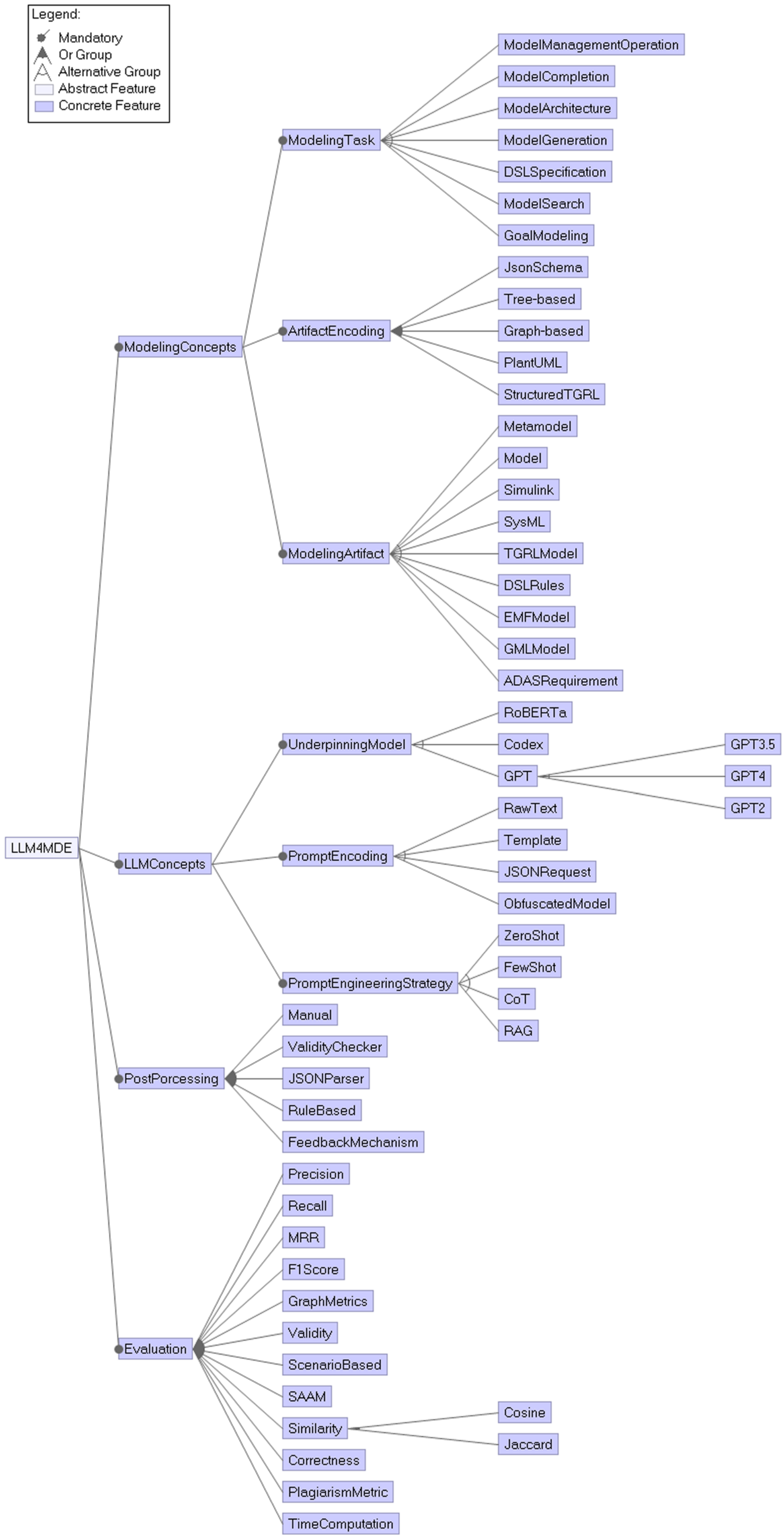


Fig. 6 Feature models representing state-of-the-art approaches employing LLMs and MDE

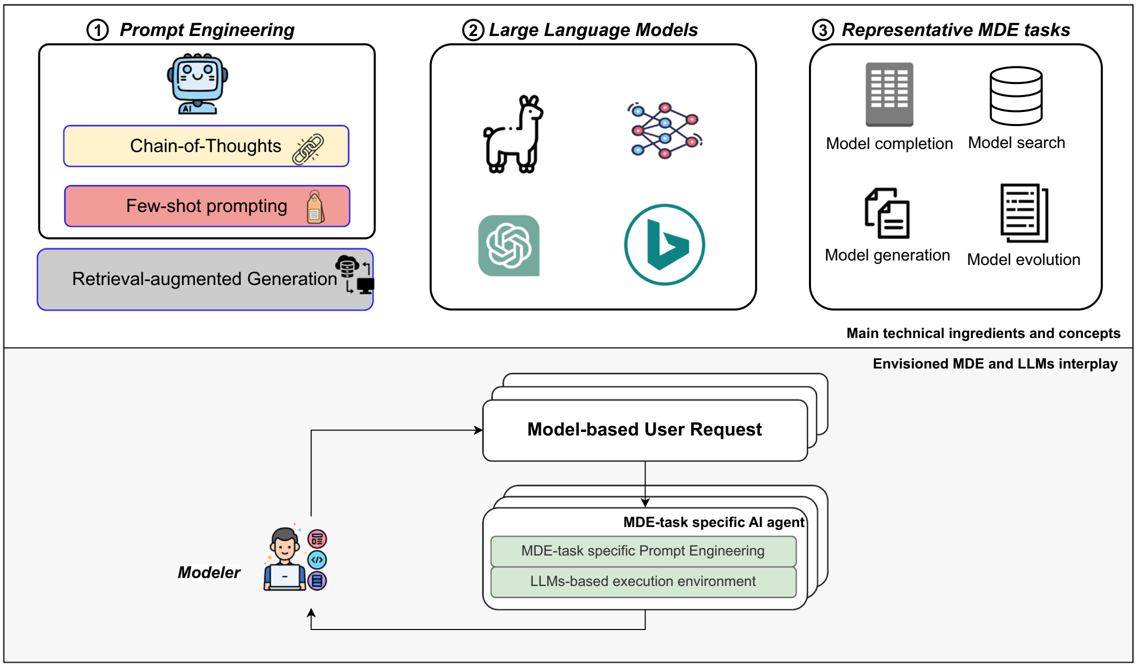


Fig. 7 Envisioned interplay of MDE and LLMs

agents fueled by LLMs. This departure from large, generalized agents, which entail resource-intensive development and training processes, offers notable advantages in terms of both time and cost-effectiveness.

In this envisioned landscape, LLMs transition from playing the role of generic execution environments to empowering MDE task-specific AI agents. These specialized agents are tailored to perform distinct model management operations, including but not limited to model evolution, model comparison, domain modeling, generation of training and testing data, and model completion. Users can increase their efficiency and relevance in development processes by aligning AI agents with specific MDE tasks. Crucially, the input to these task-specific agents is provided through specifications adhering to domain-specific languages (similarly to what was proposed by *Clarisó et al.* [73]), further enhancing the precision and applicability of the collaborative interplay between LLMs and MDE technologies. In other words, low-code environments [31] will democratize the usage of AI agents that are specific for the application domains of interests.

5.2 *LLM4MDE*: LLMs-based automation for MDE tasks

In the LLM4MDE direction, the community should continue the work initiated with automated modeling assistants, domain analysis based on neural networks, and deep learning technologies. The scope of LLM4MDE extends beyond conventional software engineering tasks. LLMs can be leveraged as tools to automate and assist in a wide range of MDE activities, from requirements elicitation to model transformation, providing valuable insights and augmenting the capabilities of modelers.

A cross-cutting concern that involves all the aspects discussed later in this section is the mitigation of hallucinations in LLMs. Hallucinations occur when LLMs produce responses that are not relevant to the input context, resulting in nonsensical, incorrect, or useless outputs. It is then crucial to address this issue to ensure the reliability and effectiveness of LLMs in MDE tasks. Various strategies, such as fine-tuning on domain-specific data, integrating additional context or constraints into the generation process, and implementing post-generation filtering techniques, can help mitigate hallucinations. Ensuring that LLMs provide accurate and contextually appropriate outputs is essential for their successful integration into MDE workflows. Moreover, designing and deploying LLMs applications with the user-centered principle will help improve the velocity and flexibility of the development process. Once the goals and needs of the system's end-users have been placed at the center of software development, they will allow developers to deliver software with appropriate usability [17].

Enhanced modeling assistance: Being built on top of automated modeling assistants and domain analysis based on neural networks, research in this direction should focus on advanced recommendation algorithms, combining LLM capabilities with existing MDE knowledge to provide accurate and context-aware suggestions for modelers. In particular, besides the research already done so far (see Sect. 4), further investigations are needed to enable personalized recommendations based on individual modeling styles and preferences. For instance, RAG techniques can be adopted to enable the generation of recommendations that not only consider the inherent characteristics of modeling artifacts, but also adapt to the specific context and preferences of the modeler.

Dealing with hallucinations: So far, various strategies have been proposed to mitigate hallucination in LLMs [85], including fine-tuning on domain-specific data, incorporating additional context or constraints into the generation process, and implementing post-generation filtering techniques to identify and remove hallucinatory responses. We do not expect to provide an exhaustive list of methods to mitigate hallucinations, rather than, we anticipate that there are at least the following ones:

– *Training and fine-tuning with high-quality datasets*: Using high-quality, well-curated training datasets reduces the chances of the model learning incorrect information. Including diverse perspectives and accurate information helps the model to provide more accurate outputs.
– *Prompt Engineering*: Designing prompts that are clear and specific can guide the model toward generating more accurate responses. Moreover, providing context within prompts will help the model understand the scope and constraints, aiming to reduce the likelihood of generating off-topic or fabricated information.
– *Users' feedback*: Incorporating feedback from users is a means to continuously improve the model's performance and reduce hallucinations over time. Moreover, allowing users to query the model iteratively and correct or clarify information can help mitigate hallucinations.

Automated model generation: This line of research is to enhance the automation capabilities within MDE by extending to the automatic generation of models or model elements based on insights from LLMs. As discussed in the previous sections, different techniques have been already proposed to synthesize models [81] or mutate existing ones according to user specified characteristics [38] e.g., to evaluate new model management tools on varying model data sets. Additionally, LLMs can assist in generating or mutating models by analyzing textual requirements and considering the specificity of the application domain of interest. Thus, for example when asked to create mutants of a given input model, instead of generating model elements named with random strings, the application domain (such as medical or industrial) will be considered to generate appropriate elements having names that make sense for the domain of interest. In this respect, prompt engineering can be used to craft effective queries or input patterns, so as to provide the model with context and constraints that steer its output toward the desired results.

Ethical and responsible use of LLMs in MDE: As LLMs become integral to MDE workflows, the ethical implications of their usage come into focus as stated by recent research [13, 37, 67, 68]. Research efforts should aim to examine issues related to bias, fairness, and transparency, proposing guidelines and best practices for the responsible integration of LLMs. The focus has to extend beyond traditional software engineering tasks, ensuring ethical considerations in diverse application domains. The relevance of such topics has been made popular, e.g., by infamous incidents in the recruitment instrument employed by Amazon[6] and the criminal recidivism predictions made by the commercial risk assessment software COMPAS [46].

Benchmarking and evaluation metrics: To facilitate the integration of LLMs into MDE workflows, researchers and practitioners can already exploit several tools and frameworks. For instance, Hugging Face Transformers[13] is a widely used library that offers access to a vast array of pretrained LLMs, such as GPT, BERT, T5, to name but a few. Essentially, these models can be fine-tuned or used directly in MDE applications. OpenAI API[14] allows access to LLMs like GPT-3 and GPT-4, enabling developers to implement these models in MDE scenarios. Over the last few years, some datasets (e.g., MAR [59], and ModelSet[15]) have been defined by the MDE community, and they can be used to further train or evaluate LLMs within the context of MDE.

To gauge the effectiveness of LLMs integration in MDE tasks, establishing standardized benchmarks becomes crucial. Metrics including model accuracy, efficiency, and adaptability go beyond traditional evaluation criteria. Moreover, qualitative metrics such as fairness, robustness should also be taken into account. In this respect, human evaluation is an essential step to validate the performance of an LLM. Such an evaluation, apart from a conventional quality measurement, can also help reveal the Helpfulness, Honesty, and Harmlessness of an LLM [10]. In fact, a manual evaluation reflects better the actual application scenario, and thus it has the potential to yield more comprehensive and accurate feedback [23]. This research direction should aim to provide a comprehensive framework for evaluating LLMs performance within the requirements of MDE scenarios. Similarly to what has been done to support a disciplined comparison of ML methods for a particular MDE task [62], there will be the need for frameworks to support the quality assessment of contents generated by LLMs and even to compare those that are produced by different models from the same queries.

[13] https://huggingface.co/docs/transformers/index

[14] https://openai.com/

[15] https://models-lab.github.io/blog/2021/modelset/

Scalability and resource efficiency: Scalability is a crucial consideration in deploying LLMs for large-scale MDE projects. Research needs to investigate methods to enhance scalability while optimizing resource utilization. Thus, techniques for deploying LLMs in resource-constrained environments must be explored, ensuring accessibility across a broad spectrum of MDE applications. Here, we need support to suggest the most energy/resource-efficient technique that can be exploited for the problem at hand. An early analysis is preferable instead of always using the most resource-consuming technologies. Of course, this can be done when the user can accept the reduced accuracy price. In other words, it is necessary to devise methodologies for early analysis and assessment of available models and techniques to identify the most suitable options for the given MDE task. Such proactive approaches will allow for informed decision-making, balancing resource constraints with the desired level of accuracy.
Security and robustness: Security is a crucial consideration when integrating LLMs within MDE workflows. It is essential to assess potential vulnerabilities and propose mitigation strategies to ensure the integrity of the model-driven ecosystem. The research direction should focus on exploring techniques to enhance the robustness of LLMs against adversarial attacks within MDE tasks. For instance, recently GitHub faced an attack that resulted in the creation of millions of code repositories containing obfuscated malware [39] These malicious repositories are clones of legitimate ones, making them challenging to distinguish. An unknown party automated a process that forks legitimate repositories, resulting in millions of forks with names identical to the originals but containing payloads wrapped under seven layers of obfuscation. Therefore, it is necessary to develop techniques and tools for detecting malicious sources when training language models for MDE tasks or any other software engineering purposes.
Long-term impact assessment: The long-term impact of adopting LLMs in MDE practices is a crucial aspect of research. Thus, it is necessary to perform studies to assess, e.g., modeler productivity, and overall product quality. Key success indicators and performance metrics have to be identified to measure sustained benefits, providing insights into the impact of LLMs integration in MDE over time.
Educational assistance: Research can be done to employing LLMs for teaching. In particular, the goal is integrating LLMs into educational environments to teach and assist new modelers in their tasks. This concept aligns with recommender systems but focuses on less granular concepts, providing educational support at a broader level. LLMs can offer explanations, context, and guidance tailored to the educational needs of novice modelers. To this end, the community might consider training LLMs using reviewed sources such as

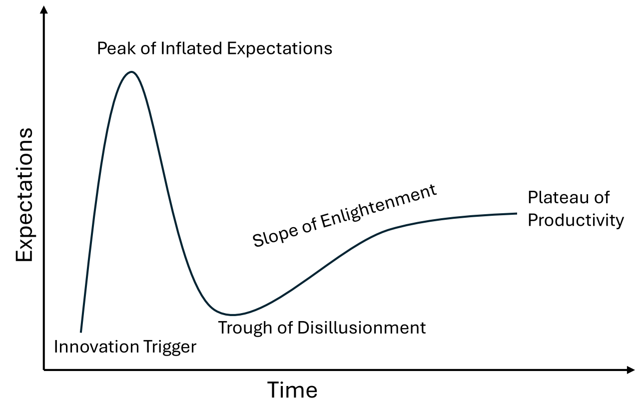


Fig. 8 Gartner Hype Cycle

SWEBOK [45], SLEBOK,[16] and thereby contribute to the development of a comprehensive model-based software engineering body of knowledge [19].
Final thoughts: The hype around using LLMs is showing their big potential in many areas. But like any new technology, the initial excitement might fade as we learn more about what they can and can not do. In particular, mirroring the Gartner Hype Cycle[17] (see Fig. 8), LLMs may undergo phases of inflated expectations, followed by a clearer understanding of their real-world applications and constraints.

We believe that LLMs represent a relevant technology to improve the *automation* aspect of model-driven engineering. However, while LLMs-based automation can benefit MDE processes, the human element remains indispensable. In particular, as discussed in Sect. 2, LLMs may produce outputs that deviate from the desired context or contain inaccuracies. Human intervention is crucial to identify and rectify such instances, ensuring the quality and reliability of model-driven artifacts. Moreover, in complex engineering tasks, particularly those involving critical systems, human oversight remains indispensable for accountability. Human-in-the-loop systems enable traceability and accountability, ensuring that decisions made by LLMs align with regulatory standards. Finally, industries such as healthcare, aerospace, and automotive adhere to stringent certification standards. Human involvement in the loop is still necessary for the validation and certification processes, ensuring compliance with regulatory requirements and safety standards.

6 Related work

In recent years, the research landscape surrounding the integration of LLMs with SE has witnessed remarkable activity.

16 https://slebok.github.io/

17 https://en.wikipedia.org/wiki/Gartner_hype_cycle

This section reviews some of the most notable studies in this topic.

A systematic mapping study [64] to analyze 248 studies from January 2010 to March 2020 reveals that the most explored SE properties of AI-based systems are dependability and safety. The study identifies various SE approaches for AI-based systems, categorized according to the SWEBOK areas, with a focus on software testing and quality, while maintenance aspects appear neglected. Data-related challenges are recurring, providing valuable insights for researchers, practitioners, and educators to understand the current state-of-the-art, address research gaps, and bridge the knowledge divide between SE and AI in curricula.

Hou *et al.* [43] focused on the application of LLMs in Software Engineering (from 2017 to 2023). Firstly, the study categorizes different LLMs used in SE tasks, describing their features and applications. Secondly, it analyzes data collection, preprocessing methods, and the importance of well-curated datasets for successful implementation. Thirdly, it investigates strategies for optimizing and evaluating LLM performance in SE. Lastly, it examines specific SE tasks where LLMs have demonstrated success, highlighting their practical contributions. The review aims to provide a comprehensive understanding of the current state-of-the-art, identify research gaps, and suggest promising areas for future study in the intersection of LLMs and SE.

Similarly, Fan *et al.* [36] presented a survey on LLMs in software engineering and highlights the research challenges in applying LLMs to address technical issues faced by software engineers. LLMs are known for their innovative and generative capabilities, which have a significant impact on various SE activities such as coding, design, requirements, repair, refactoring, performance improvement, documentation, and analytics. However, the emergence of these properties also poses significant challenges in accurately identifying solutions and addressing issues like hallucinations. The survey emphasizes the importance of hybrid techniques that combine traditional SE approaches with LLMs to ensure the development and deployment of reliable, efficient, and effective LLM-based SE solutions.

In the MDE community, Combemale *et al.* [27] discussed how a large language model like ChatGPT can be used in software development, particularly in creating models that represent software systems. The authors explored different scenarios, from fully automated generation of code from requirements to using ChatGPT as an assistant for human modelers. They acknowledged the challenges of ensuring reliable and trustworthy results from AI-generated models and the need for large libraries of existing models for ChatGPT to learn from.

Di Ruscio *et al.* [32] elaborated on the use of model-driven engineering and machine learning techniques to support the management of modeling ecosystems. The paper identifies and discusses possible lines of research to explore the adoption of existing machine learning techniques to enhance the management of modeling ecosystems.

In a recent paper [21], the authors elaborated on the potential of LLMs, such as Copilot and ChatGPT, to revolutionize software development. The paper examines the current capabilities of ChatGPT for modeling tasks and assisting modelers, and identifies several shortcomings, including syntactic and semantic deficiencies, lack of consistency in responses, and scalability issues. The paper provides suggestions on how the modeling community can help improve the current capabilities of ChatGPT and future LLMs for software modeling.

7 Conclusion and future work

In contrast to broader research on large language models (LLMs) in software engineering, our paper focuses on the specific synergy between LLMs and model-driven engineering. We explored how LLMs automate tasks unique to MDE, like model repository classification and advanced model recommenders. The paper also outlines the technical considerations for seamlessly integrating LLMs into MDE workflows, offering a practical guide for researchers and practitioners. This paper proposed also a targeted research agenda, identifying challenges and opportunities for leveraging LLMs in MDE and vice versa. This roadmap contributes to evolving MDE practices and offers a forward-looking perspective on the transformative role of large language models in software engineering and model-driven practices.

Acknowledgements This work was partially supported by the following Italian research projects: EMELIOT (PRIN 2020, grant n. 2020W3A5FY) and TRex SE (PRIN 2022, grant n. 2022LKJWHC). Additional support was provided by the European Union NextGenerationEU through the Italian Ministry of University and Research for the MATTERS project, funded under the cascade scheme of the SERICS program (CUP J33C22002810001), Spoke 8, within the Italian PNRR Mission 4, Component 2, as well as the FRINGE project (PRIN 2022 PNRR, grant n. P2022553SL).

References

1. Abukhalaf, S., Hamdaqa, M., Khomh, F.: On codex prompt engineering for ocl generation: An empirical study. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 148–157 (2023). doi:https://doi.org/10.1109/MSR59073.2023.00033
2. Abukhalaf, S., Hamdaqa, M., Khomh, F.: Pathocl: Path-based prompt augmentation for ocl generation with gpt-4. In: Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering, FORGE '24, p. 108-118. Association for Computing Machinery, New York, NY, USA (2024). doi:https://doi.org/10.1145/3650105.3652290

3. Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M.S., Mikkonen, T.: Towards Human-Bot Collaborative Software Architecting with ChatGPT. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE '23, pp. 279–285. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3593434.3593468. Read_Status: New Read_Status_Date: 2024-07-17T07:38:55.084Z
4. Ahmed, T., Devanbu, P.: Few-shot training llms for project-specific code-summarization. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22. Association for Computing Machinery, New York, NY, USA (2023). doi:https://doi.org/10.1145/3551349.3559555
5. Amatriain, X.: Prompt design and engineering: Introduction and advanced methods (2024). https://arxiv.org/abs/2401.14423
6. Amazon: Amazon's ai hiring software showed bias against women. https://www.theguardian.com/technology/2018/oct/10/amazon-hiring-ai-gender-bias-recruiting-engine (2018). Accessed: 2024-07-22
7. Apvrille., L., Sultan., B.: System architects are not alone anymore: Automatic system modeling with ai. In: Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD, pp. 27–38. INSTICC, SciTePress (2024). doi:https://doi.org/10.5220/0012320100003645
8. Ardimento, P., Bernardi, M.L., Cimitile, M.: Teaching uml using a rag-based llm. In: 2024 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2024). doi:https://doi.org/10.1109/IJCNN60899.2024.10651492
9. Arulmohan, S., Meurs, M.J., Mosser, S.: Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 580–587. IEEE, Västerås, Sweden (2023). doi:https://doi.org/10.1109/MODELS-C59198.2023.00096. https://ieeexplore.ieee.org/document/10350787/
10. Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., Dassarma, N., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Kernion, J., Ndousse, K., Olsson, C., Amodei, D., Brown, T.B., Clark, J., McCandlish, S., Olah, C., Kaplan, J.: A general language assistant as a laboratory for alignment. ArXiv **abs/2112.00861** (2021). https://api.semanticscholar.org/CorpusID:244799619
11. Banerjee, S., Lavie, A.: Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, pp. 65–72 (2005)
12. Bano, M., Hoda, R., Zowghi, D., Treude, C.: Large language models for qualitative research in software engineering: exploring opportunities and challenges. Autom. Softw. Eng. **31**(1), 8 (2024). https://doi.org/10.1007/S10515-023-00407-8
13. Basta, C., Costa-jussà, M.R., Casas, N.: Evaluating the underlying gender bias in contextualized word embeddings. In: M.R. Costa-jussà, C. Hardmeier, W. Radford, K. Webster (eds.) Proceedings of the First Workshop on Gender Bias in Natural Language Processing, pp. 33–39. Association for Computational Linguistics, Florence, Italy (2019). doi:https://doi.org/10.18653/v1/W19-3805. https://aclanthology.org/W19-3805
14. Belzner, L., Gabor, T., Wirsing, M.: Large language model assisted software engineering: Prospects, challenges, and a case study. In: Steffen, B. (ed.) Bridging the Gap Between AI and Reality, pp. 355–374. Springer Nature Switzerland, Cham (2024)
15. Bertram, V., Boß, M., Kusmenko, E., Nachmann, I.H., Rumpe, B., Trotta, D., Wachtmeister, L.: Neural language models and few shot learning for systematic requirements processing in mdse. In: Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2022, p. 260-265. Association for Computing Machinery, New York, NY, USA (2022). doi:https://doi.org/10.1145/3567512.3567534
16. Bézivin, J., Jouault, F., Valduriez, P.: On the need for megamodels. In: proceedings of the OOPSLA/GPCE: best practices for model-driven software development workshop, 19th Annual ACM conference on object-oriented programming, systems, languages, and applications, pp. 1–9. Citeseer (2004)
17. Brhel, M., Meth, H., Maedche, A., Werder, K.: Exploring principles of user-centered agile software development: A literature review. Information and Software Technology **61**, 163–181 (2015). doi:10.1016/j.infsof.2015.01.004. https://www.sciencedirect.com/science/article/pii/S0950584915000129
18. Bucaioni, A., Ekedahl, H., Helander, V., Nguyen, P.T.: Programming with ChatGPT: How far can we go? Machine Learning with Applications **15**, 100526 (2024). doi:10.1016/j.mlwa.2024.100526. https://www.sciencedirect.com/science/article/pii/S2666827024000021
19. Burgueño, L., Ciccozzi, F., Famelis, M., Kappel, G., Lambers, L., Mosser, S., Paige, R.F., Pierantonio, A., Rensink, A., Salay, R., Taentzer, G., Vallecillo, A., Wimmer, M.: Contents for a model-based software engineering body of knowledge. Softw. Syst. Model. **18**(6), 3193–3205 (2019). https://doi.org/10.1007/s10270-019-00746-9
20. Callison-Burch, C., Osborne, M., Koehn, P.: Re-evaluating the role of bleu in machine translation research. In: 11th conference of the european chapter of the association for computational linguistics, pp. 249–256 (2006)
21. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. Softw. Syst. Model. **22**(3), 781–793 (2023). https://doi.org/10.1007/s10270-023-01105-5
22. Chaaben, M.B., Burgueño, L., Sahraoui, H.: Towards Using Few-Shot Prompt Learning for Automating Model Completion. In: Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER '23, pp. 7–12. IEEE, IEEE Press, Melbourne, Australia (2023). doi:https://doi.org/10.1109/ICSE-NIER58687.2023.00008
23. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., Xie, X.: A survey on evaluation of large language models. Intell. Syst. Technol, ACM Trans (2024). https://doi.org/10.1145/3641289
24. Chen, B., Chen, K., Hassani, S., Yang, Y., Amyot, D., Lessard, L., Mussbacher, G., Sabetzadeh, M., Varró, D.: On the use of gpt-4 for creating goal models: An exploratory study. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), pp. 262–271 (2023). doi:https://doi.org/10.1109/REW57809.2023.00052
25. Chen, K., Yang, Y., Chen, B., Hernández López, J.A., Mussbacher, G., et al.: Automated Domain Modeling with Large Language Models: A Comparative Study. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 162–172 (2023). doi:https://doi.org/10.1109/MODELS58315.2023.00037. https://ieeexplore.ieee.org/abstract/document/10344012
26. Chen, X., Liu, T., Fournier-Viger, P., Zhang, B., Long, G., Zhang, Q.: A fine-grained self-adapting prompt learning approach for few-shot learning with pre-trained language models. Knowledge-Based Systems **299**, 111968 (2024). doi:https://doi.org/10.1016/j.knosys.2024.111968. https://www.sciencedirect.com/science/article/pii/S0950705124006026
27. Combemale, B., Gray, J., Rumpe, B.: ChatGPT in software modeling. Softw. Syst. Model **22**(3), 777–779 (2023). https://doi.org/10.1007/s10270-023-01106-4

28. Darif, I., Politowski, C., El Boussaidi, G., Benzarti, I., Kpodjedo, S.: A model-driven and template-based approach for requirements specification. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 239–249. IEEE (2023)
29. Deeptimahanti, D.K., Babar, M.A.: An automated tool for generating uml models from natural language requirements. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 680–682. IEEE (2009)
30. Di Rocco, J., Di Ruscio, D., Härtel, J., Iovino, L., Lämmel, R., Pierantonio, A.: Understanding mde projects: megamodels to the rescue for architecture recovery. Softw. Syst. Model. **19**, 401–423 (2020)
31. Di Ruscio, D., Kolovos, D.S., de Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: Two sides of the same coin? Softw. Syst. Model. **21**(2), 437–446 (2022). https://doi.org/10.1007/s10270-021-00970-2
32. Di Ruscio, D., Nguyen, P.T., Pierantonio, A.: Machine Learning for Managing Modeling Ecosystems: Techniques, Applications, and a Research Vision. Springer International Publishing, Cham (2023). https://doi.org/10.1007/978-3-031-36060-2_10
33. Doan, T.T.H., Nguyen, P.T., Di Rocco, J., Di Ruscio, D.: Too long; didn't read: Automatic summarization of github readme.md with transformers. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE '23, p. 267-272. Association for Computing Machinery, New York, NY, USA (2023). doi:https://doi.org/10.1145/3593434.3593448
34. Dobrica, L., Niemelä, E.: A survey on software architecture analysis methods. IEEE Trans. Softw. Eng. **28**(7), 638–653 (2002). https://doi.org/10.1109/TSE.2002.1019479
35. Elallaoui, M., Nafil, K., Touahni, R.: Automatic transformation of user stories into uml use case diagrams using nlp techniques. Proc. Comput. sci. **130**, 42–49 (2018)
36. Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., Zhang, J.M.: Large language models for software engineering: Survey and open problems arXiv:2310.03533 (2023). http://arxiv.org/abs/2310.03533. ArXiv:2310.03533 [cs]
37. Gehman, S., Gururangan, S., Sap, M., Choi, Y., Smith, N.A.: RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In: T. Cohn, Y. He, Y. Liu (eds.) Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 3356–3369. Association for Computational Linguistics, Online (2020). doi:https://doi.org/10.18653/v1/2020.findings-emnlp.301
38. Gómez-Abajo, P., Guerra, E., De Lara, J.: Wodel: a domain-specific language for model mutation. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1968–1973. ACM, Pisa Italy (2016). doi:https://doi.org/10.1145/2851613.2851751
39. Goodin, D.: Github besieged by millions of malicious repositories in ongoing attack. https://arstechnica.com/security/2024/02/github-besieged-by-millions-of-malicious-repositories-in-ongoing-attack/ (2024). Accessed: 2024-07-22
40. Goswami, J., Prajapati, K.K., Saha, A., Saha, A.K.: Parameter-efficient fine-tuning large language model approach for hospital discharge paper summarization. Applied Soft Computing **157**, 111531 (2024). doi:https://doi.org/10.1016/j.asoc.2024.111531. https://www.sciencedirect.com/science/article/pii/S1568494624003053
41. Gulia, S., Choudhury, T.: An efficient automated design to generate uml diagram from natural language specifications. In: 2016 6th international conference-cloud system and big data engineering (Confluence), pp. 641–648. IEEE (2016)
42. Henrickson, L., Meroño-Peñuela, A.: Prompting meaning: a hermeneutic approach to optimising prompt engineering with chatgpt. AI and Society (2023). doi:10.1007/s00146-023-01752-8. Publisher Copyright: © 2023, The Author(s)
43. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H.: Large language models for software engineering: A systematic literature review arXiv:2308.10620 (2024). doi:https://doi.org/10.48550/arXiv.2308.10620. http://arxiv.org/abs/2308.10620. ArXiv:2308.10620 [cs]
44. Huang, L., Duan, Y., Sun, X., Lin, Z., Zhu, C.: Enhancing uml class diagram abstraction with knowledge graph. In: Intelligent Data Engineering and Automated Learning–IDEAL 2016: 17th International Conference, Yangzhou, China, October 12–14, 2016, Proceedings 17, pp. 606–616. Springer (2016)
45. IEEE: Software engineering body of knowledge. https://www.computer.org/education/bodies-of-knowledge/software-engineering (2024). Accessed: 2024-07-22
46. Jeff Larson Surya Mattu, L.K., Angwin, J.: How we analyzed the compas recidivism algorithm. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm (2016). Accessed: 2024-07-22
47. Jin, H., Huang, L., Cai, H., Yan, J., Li, B., Chen, H.: From llms to llm-based agents for software engineering: A survey of current, challenges and future (2024). https://arxiv.org/abs/2408.02479
48. Kareem, W., Abbas, N.: Fighting lies with intelligence: Using large language models and chain of thoughts technique to combat fake news. In: Bramer, M., Stahl, F. (eds.) Artificial Intelligence XL, pp. 253–258. Springer Nature Switzerland, Cham (2023)
49. Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33**(2004), 1–26 (2004)
50. Kulkarni, V., Reddy, S., Barat, S., Dutta, J.: Toward a Symbiotic Approach Leveraging Generative AI for Model Driven Engineering. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 184–193 (2023). doi:https://doi.org/10.1109/MODELS58315.2023.00039. https://ieeexplore-ieee-org.univaq.idm.oclc.org/document/10343767/?arnumber=10343767. Read_Status: New Read_Status_Date: 2024-07-23T09:49:31.511Z
51. Lee, G.G., Latif, E., Wu, X., Liu, N., Zhai, X.: Applying large language models and chain-of-thought for automatic scoring. Computers and Education: Artificial Intelligence **6**, 100213 (2024). doi:https://doi.org/10.1016/j.caeai.2024.100213. https://www.sciencedirect.com/science/article/pii/S2666920X24000146
52. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20. Curran Associates Inc., Red Hook, NY, USA (2020)
53. Li, X., Yuan, S., Gu, X., Chen, Y., Shen, B.: Few-shot code translation via task-adapted prompt learning. Journal of Systems and Software **212**, 112002 (2024). doi:https://doi.org/10.1016/j.jss.2024.112002. https://www.sciencedirect.com/science/article/pii/S0164121224000451
54. Lin, B.: Reinforcement learning and bandits for speech and language processing: Tutorial, review and outlook. Expert Systems with Applications **238**, 122254 (2024). doi:https://doi.org/10.1016/j.eswa.2023.122254. https://www.sciencedirect.com/science/article/pii/S0957417423027562
55. Lin, C.Y.: ROUGE: A package for automatic evaluation of summaries. In: Text Summarization Branches Out, pp. 74–81. Association for Computational Linguistics, Barcelona, Spain (2004). https://aclanthology.org/W04-1013
56. Liu, X., Pang, T., Fan, C.: Federated prompting and chain-of-thought reasoning for improving llms answering. In: Jin, Z., Jiang, Y., Buchmann, R.A., Bi, Y., Ghiran, A.M., Ma, W. (eds.) Knowledge Science, Engineering and Management, pp. 3–11. Springer Nature Switzerland, Cham (2023)
57. Liu, Y., Le-Cong, T., Widyasari, R., Tantithamthavorn, C., Li, L., Le, X.B.D., Lo, D.: Refining chatgpt-generated code: Char-

acterizing and mitigating code quality issues. ACM Trans. Softw. Eng. Methodol. (2024). doi:https://doi.org/10.1145/3643674. Just Accepted
58. López, J.A.H., Cánovas Izquierdo, J.L., Cuadrado, J.S.: Modelset: a dataset for machine learning in model-driven engineering. Software and Systems Modeling pp. 1–20 (2021)
59. López, J.A.H., Cuadrado, J.S.: Mar: a structure-based search engine for models. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems, pp. 57–67 (2020)
60. López, J.A.H., Durá, C., Cuadrado, J.S.: Word embeddings for model-driven engineering. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 151–161. IEEE (2023)
61. López, J.A.H., Izquierdo, J.L.C., Cuadrado, J.S.: Using the modelset dataset to support machine learning in model-driven engineering. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 66–70 (2022)
62. López, J.A.H., Rubei, R., Cuadrado, J.S., Ruscio, D.D.: Machine learning methods for model classification: a comparative study. In: E. Syriani, H.A. Sahraoui, N. Bencomo, M. Wimmer (eds.) Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022, pp. 165–175. ACM (2022). doi:https://doi.org/10.1145/3550355.3552461
63. Lundell, B., Lings, B., Persson, A., Mattsson, A.: Uml model interchange in heterogeneous tool environments: An analysis of adoptions of xmi 2. In: Model Driven Engineering Languages and Systems: 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006. Proceedings 9, pp. 619–630. Springer (2006)
64. Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M., Wagner, S.: Software engineering for ai-based systems: A survey. ACM Transactions on Software Engineering and Methodology **31**(2), 1–59 (2022). https://doi.org/10.1145/3487043
65. Martino, A., Iannelli, M., Truong, C.: Knowledge injection to counter large language model (llm) hallucination. In: Pesquita, C., Skaf-Molli, H., Efthymiou, V., Kirrane, S., Ngonga, A., Collarana, D., Cerqueira, R., Alam, M., Trojahn, C., Hertling, S. (eds.) The Semantic Web: ESWC 2023 Satellite Events, pp. 182–185. Springer Nature Switzerland, Cham (2023)
66. Mastropaolo, A., Ciniselli, M., Penta, M.D., Bavota, G.: Evaluating code summarization techniques: A new metric and an empirical characterization (2023)
67. Morales, S., Clarisó, R., Cabot, J.: Automating bias testing of llms. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1705–1707 (2023). doi:https://doi.org/10.1109/ASE56229.2023.00018
68. Nemani, P., Joel, Y.D., Vijay, P., Liza, F.F.: Gender bias in transformers: A comprehensive review of detection and mitigation strategies. Natural Language Processing Journal **6**, 100047 (2024). doi:https://doi.org/10.1016/j.nlp.2023.100047. https://www.sciencedirect.com/science/article/pii/S2949719123000444
69. Nguyen, P.T., Di Rocco, J., Di Sipio, C., Rubei, R., Di Ruscio, D., Di Penta, M.: GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. Journal of Systems and Software **214**, 112059 (2024). doi:https://doi.org/10.1016/j.jss.2024.112059. https://www.sciencedirect.com/science/article/pii/S0164121224001043
70. Ozkaya, I.: Application of large language models to software engineering tasks: Opportunities, risks, and implications. IEEE Software **40**(3), 4–8 (2023). https://doi.org/10.1109/MS.2023.3248401
71. Ozkaya, I.: The next frontier in software development: Ai-augmented software development processes. IEEE Software **40**(4), 4–9 (2023). https://doi.org/10.1109/MS.2023.3278056
72. Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., Shoham, Y.: In-Context Retrieval-Augmented Language Models. Transactions of the Association for Computational Linguistics **11**, 1316–1331 (2023). https://doi.org/10.1162/tacl_a_00605
73. Robert, C., Jordi, C.: Model-driven prompt engineering. In: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (2023)
74. Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., et al.: Code llama: Open foundation models for code (2023)
75. Sauvola, J., Tarkoma, S., Klemettinen, M., Riekki, J., Doermann, D.: Future of software development with generative ai. Automated Software Engineering **31** (2024). doi:https://doi.org/10.1007/s10515-024-00426-z
76. Sağlam, T., Hahner, S., Schmid, L., Burger, E.: Automated Detection of AI-Obfuscated Plagiarism in Modeling Assignments. In: Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET '24, pp. 297–308. Association for Computing Machinery, New York, NY, USA (2024). doi:https://doi.org/10.1145/3639474.3640084. Read_Status: New Read_Status_Date: 2024-07-22T13:08:19.395Z
77. Schmidt, D.: Guest editor's introduction: Model-driven engineering. Computer **39**(2), 25–31 (2006). https://doi.org/10.1109/MC.2006.58
78. Shrestha, S.L.: Harnessing Large Language Models for Simulink Toolchain Testing and Developing Diverse Open-Source Corpora of Simulink Models for Metric and Evolution Analysis. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, pp. 1541–1545. Association for Computing Machinery, New York, NY, USA (2023). doi:https://doi.org/10.1145/3597926.3605233
79. Shrestha, S.L., Csallner, C.: SLGPT: Using Transfer Learning to Directly Generate Simulink Model Files and Find Bugs in the Simulink Toolchain. In: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, EASE '21, pp. 260–265. Association for Computing Machinery, New York, NY, USA (2021). doi:https://doi.org/10.1145/3463274.3463806
80. Taulli, T.: Prompt Engineering, pp. 51–64. Apress, Berkeley, CA (2023). doi:https://doi.org/10.1007/978-1-4842-9852-7_4
81. Varró, D., Semeráth, O., Szárnyas, G., Horváth, Á.: Towards the automated generation of consistent, diverse, scalable and realistic graph models. In: Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, pp. 285–312. Springer (2018)
82. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., et al.: Attention is all you need. In: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017). https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
83. Wang, C., Yang, Y., Gao, C., Peng, Y., Zhang, H., et al.: Prompt tuning in code intelligence: An experimental evaluation. IEEE Transactions on Software Engineering **01**, 1–17 (2023). https://doi.org/10.1109/TSE.2023.3313881
84. Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., Wang, Q.: Software testing with large language models: Survey, landscape, and vision. IEEE Trans. Softw. Eng. (2024). https://doi.org/10.1109/TSE.2024.3368208
85. Wang, L., He, J., Li, S., Liu, N., Lim, E.P.: Mitigating fine-grained hallucination by fine-tuning large vision-language models with caption rewrites. In: Rudinac, S., Hanjalic, A., Liem, C., Worring,

M., Jónsson, B.A., Liu, B., Yamakata, Y. (eds.) MultiMedia Modeling, pp. 32–45. Springer Nature Switzerland, Cham (2024)
86. Wang, L., Lepage, Y.: Learning from masked analogies between sentences at multiple levels of formality. Annals Math. Artif. Intell. (2023). https://doi.org/10.1007/s10472-023-09918-2
87. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models (2023). https://arxiv.org/abs/2203.11171
88. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022)
89. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Software and Systems Modeling pp. 1–19 (2022)
90. Yang, S., Zhu, J., Wang, J., Xu, X., Shao, Z., Yao, L., Zheng, B., Huang, H.: Retrieval-augmented generation with quantized large language models: A comparative analysis. In: Proceedings of the 2023 5th International Conference on Internet of Things, Automation and Artificial Intelligence, IoTAAI '23, p. 120-124. Association for Computing Machinery, New York, NY, USA (2024). doi:https://doi.org/10.1145/3653081.3653102
91. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T.L., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models (2023). https://arxiv.org/abs/2305.10601
92. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2022)
93. Zhang, Y., Qiu, Z., Stol, K.J., Zhu, W., Zhu, J., Tian, Y., Liu, H.: Automatic commit message generation: A critical review and directions for future work. IEEE Trans. on Softw. Eng. (2024). https://doi.org/10.1109/TSE.2024.3364675
94. Zhou, X., Yang, L., Wang, X., Zhan, H., Sun, R.: Two stages prompting for few-shot multi-intent detection. Neurocomputing **579**, 127424 (2024). doi:https://doi.org/10.1016/j.neucom.2024.127424. https://www.sciencedirect.com/science/article/pii/S0925231224001954
95. Zhu, Y., Li, J., Li, G., Zhao, Y., Jin, Z., Mei, H.: Hot or cold? adaptive temperature sampling for code generation with large language models. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 437–445 (2024)
96. Zielińska, A., Organisciak, P., Dumas, D., Karwowski, M.: Lost in translation? not for large language models: Automated divergent thinking scoring performance translates to non-english contexts. Thinking Skills and Creativity **50**, 101414 (2023). https://doi.org/10.1016/j.tsc.2023.101414. https://www.sciencedirect.com/science/article/pii/S1871187123001827

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Juri Di Rocco is an assistant professor at the University of L'Aquila, Italy. He obtained a PhD in Computer Science from the University of L'Aquila. He is interested in several aspects of software language engineering and model-driven engineering (MDE) including domain-specific modeling languages, model transformation, model differencing, modeling repositories and mining techniques. More information is available at http://www.di.univaq.it/juri.dirocco.

Contact him at juri.dirocco@univaq.it

Davide Di Ruscio is a Professor at the Department of Information Engineering Computer Science and Mathematics of the University of L'Aquila. His main research interests are related to several aspects of software engineering and model-driven engineering (MDE), including domain-specific modeling languages, model differencing, coupled evolution, and recommendation systems. Davide is on the editorial board of the International Journal on Software and Systems Modeling (SoSyM), of IEEE Software, of the Journal of Object Technology, and of the Business and Information Systems Engineering journal, and he is a regular reviewer of many journals including IEEE Transactions on Software Engineering, Science of Computer Programming, Software and Systems Modeling, and Journal of Systems and Software. He has served in the organization and program committees of more than 100 international events, including MODELS, STAF, ICSE, FSE, EASE, and SANER. Web: https://www.disim.univaq.it/DavideDiRuscio

Claudio Di Sipio is a postdoc researcher in the Department of Engineering, Mathematics, and Computer Science at the University of L'Aquila. He took his PhD in 2023 at the University of L'Aquila, and he has been an invited researcher at GEODES lab, University of Montreal, for six months. His research interests include recommendation systems for software engineering, mining OSS repositories, model-driven engineering, and application of ML/AI techniques for software engineering. He served as a program committee member for different international conferences, including ASE, MSR, FORGE, and MODELS. In addition, he served as a reviewer for the IEEE Transactions on Software Engineering (TSE) and the Springer Software and Systems Modeling (SoSyM) journals. Further information about him is available at https://claudiodsi.github.io/. Contact him at claudio.disipio@univaq.it.

Phuong T. Nguyen is a associate professor at the University of L'Aquila, Italy. He obtained a PhD in Computer Science from the University of Jena, Germany. Since graduation, he has been a university teaching and research assistant in Vietnam and Italy. His research interests include Computer Networks, Semantic Web, Recommender Systems, and Machine Learning. Phuong is an associate editor of Springer Applied Intelligence, and he is on the editorial board of Springer Software Quality Journal. Moreover, he has been a reviewer for many premier journals and conferences including TSE, TOSEM, JSS, AAAI, FSE, SANER, ESEM, and EASE. More information is available at the following address: https://www.disim.univaq.it/ThanhPhuong

Riccardo Rubei is a postdoctoral researcher at the University of L'Aquila(Italy). He earned his PhD in 2022 from the University of L'Aquila. His research interest is related to software engineering, recommender systems and several aspects of MDE. Furthermore, he is active in the field of sustainability and green software engineering. He is organizing the workshops in STAF 2024 and 2025 entitled "Large Language Models for Model-Driven Engineering" and "Foundations and Practice of Visual Modeling" in Models 2024. Contact him at riccardo.rubei@univaq.it