Model Generation with LLMs: from Requirements to UML Sequence Diagrams

Sallam Abualhaija*, Chetan Arora†, Alessio Ferrari‡
* SnT University of Luxembourg, Luxembourg, Email: sallam.abualhaija@uni.lu
† Monash University, Email: chetan.arora@monash.edu
‡ Consiglio Nazionale delle Ricerche (CNR), Email: alessio.ferrari@isti.cnr.it

Abstract—**Complementing natural language (NL) requirements with graphical models can improve stakeholders' communication and provide directions for system design. However, creating models from requirements involves manual effort. The advent of generative large language models (LLMs), ChatGPT being a notable example, offers promising avenues for automated assistance in model generation. This paper investigates the capability of ChatGPT to generate a specific type of model, i.e., UML *sequence diagrams*, from NL requirements. We conduct a qualitative study in which we examine the sequence diagrams generated by ChatGPT for 28 requirements documents of various types and from different domains. Observations from the analysis of the generated diagrams have systematically been captured through evaluation logs, and categorized through thematic analysis. Our results indicate that, although the models generally conform to the standard and exhibit a reasonable level of understandability, their completeness and correctness with respect to the specified requirements often presents challenges. This issue is particularly pronounced in the presence of requirements smells, such as ambiguity and inconsistency. The insights derived from this study can influence the practical utilization of LLMs in the RE process, and open the door to novel RE-specific prompting strategies targeting effective model generation.**

I. Introduction

Graphical models are recognized to be an effective tool for facilitating communication between different stakeholders involved in the requirements engineering (RE) process and guiding towards the design of a system [1]. However, requirements are typically written in natural language (NL) [2], and complementing them with models requires significant manual effort [3]. The support of natural language processing (NLP) tools for model generation can greatly facilitate the work of requirements engineers and streamline the RE processes [4].

RE models can use various graphical notations, including well-known goal-oriented RE notations [5], along with commonly adopted representations such as Unified Modeling Language (UML) diagrams [6]. UML is a widely known and used semi-formal language for software design and requirements modelling [2], which includes structural models, e.g., class diagrams, and behavioral ones, e.g., sequence and activity diagrams.

Existing work in RE for automatically generating UML diagrams from requirements typically use heuristic rule-based NLP approaches [3], [7], [8]. Such approaches have several limitations, including significant manual effort for construction and maintenance, and difficult adaptability to different contexts [9]. With the recent advances in NLP technologies in general and generative large language models (LLMs) in particular, it becomes possible to overcome some of these limitations [10]. Generative LLMs exhibit acceptable contextual understanding, are typically pre-trained, and can be used out-of-the-box, thus reducing human effort in building a model generation tool.

This paper aims to examine the capability of ChatGPT, a well-known LLM, to generate UML *sequence diagrams*. Sequence diagrams are behavioral models that represent interactions between different components of the system-to-be [6]. They are particularly useful, as they are able to represent the dynamic behavior of a system, which is complementary to the structural (static) view given by class diagrams. Our motivation for focusing on sequence diagrams is that, despite being arguably simple to understand by different stakeholders, they are less studied in the RE literature [11]. Previous studies exploring the problems of using LLMs for model generation consider goal models [12] or class diagrams [13], [14], but, to our knowledge, none of the existing studies target sequence diagrams.

To study the model generation capability of ChatGPT, we performed an exploratory study involving three researchers—the authors of this paper—who have 7 to 13 years of experience in both RE and NLP, and practical confidence with sequence diagrams. Specifically, two researchers prompted ChatGPT to generate sequence diagrams corresponding to 28 NL requirements documents covering requirements specified in different formats, including "shall"-style requirements, user stories, and use case specifications. The researchers also introduced several variants of the same requirements to simulate realistic scenarios, hence exposing ChatGPT to common challenges. Examples of such challenges include the presence of smells or the evolution of requirements, i.e., addition, removal, modifications. The researchers *scored* the quality of the diagrams according to different criteria and tracked their observations in structured evaluation logs, one for each generated diagram. The observations were oriented to highlight quality *issues* in the diagrams. Following this, the third researcher performed a thematic analysis [15] on the evaluation logs and identified 23 main categories of issues. From the results, it emerges that the generated diagrams score well for understandability, standard compliance, and terminological alignment with the requirements. However, they exhibit significant issues related to completeness and correctness, such as missing or incorrect elements, structural issues, or components that deviate from what is specified by the requirements. These issues become more evident in the presence of low-quality requirements that include ambiguities or inconsistencies, and when technical/contextual knowledge is needed to correctly interpret the requirements.

Contributions. Our study contributes with a structured framework of issues associated with the generation of sequence diagrams from NL requirements through ChatGPT. Our results outline possible avenues for future research. These include the need for iterative, RE-specific prompting solutions, as well as the need to address tacit/domain knowledge issues that affect a general-purpose LLM such as ChatGPT, when dealing with technical requirements data.

II. Background and Related Work

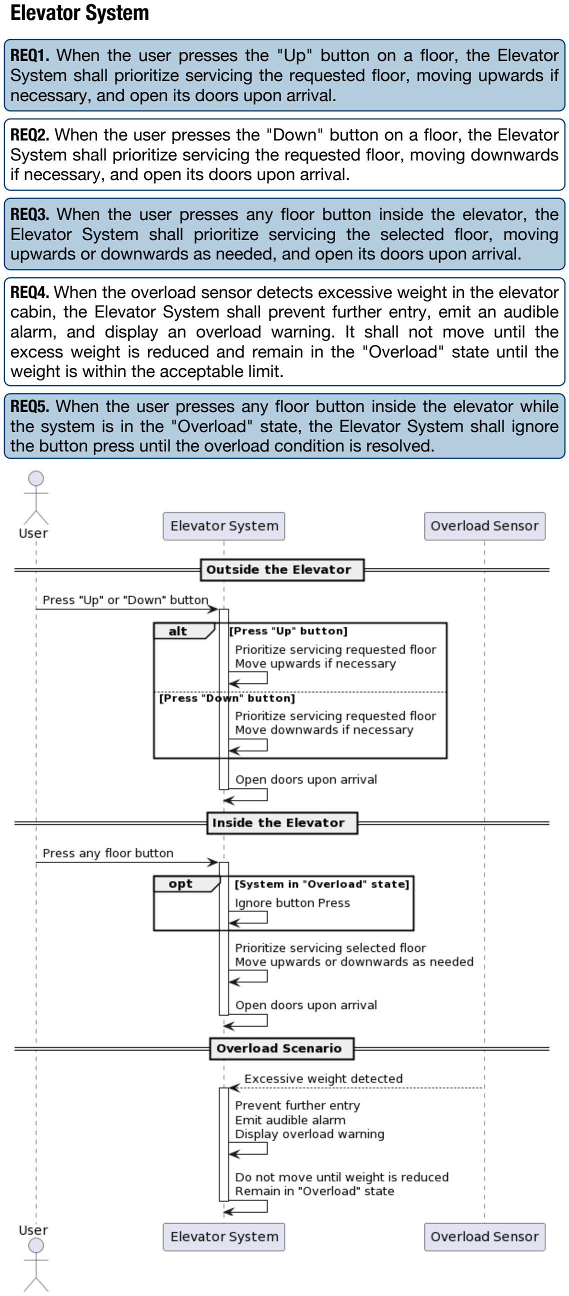
A. Sequence Diagrams

UML sequence diagrams are models representing interactions between different system components in terms of function calls and messages [6]. Fig. 1 shows an example of requirements for an elevator system and the corresponding sequence diagram. The user is represented with a stylized figure, while system components are represented as rectangles at the top and at the bottom of the diagram. In the figure, we see the User and two components: the Elevator System and the Overload Sensor. Components are associated with vertical lifelines, while the horizontal arrows identify function calls (solid line) and messages (dashed line). The models can also include alternative choices (**alt**, e.g., Press “Up” (“Down”) button), and optional steps (**opt**, e.g., System in “Overload” state), enclosed in boxes. The syntax also allows separation into conceptual blocks, e.g., “Outside the Elevator”, identified by horizontal lines. The given example is a highly simplified case, which we use to introduce the syntax of sequence diagrams. In our study, we use more complex, real-world requirements cases.

We note that contrary to formal sequence diagrams, where the labels on the arrows include pseudo-code, we generate abstract diagrams, where the labels are free-form text. The reason is that our analysis aims to explore models for facilitating interaction with stakeholders who may not have programming experience.

B. Related Work

Model Generation. Model generation is a key task in NLP for RE. This involves creating model abstractions—typically in a graphical form—from input requirements. The systematic mapping study from Zhao *et al.* [16], which analyses studies in NLP for RE from 1983 until 2019, reports 59 contributions on model generation, with the majority of automated NLP tools targeting this task. Model generation can take different flavors, from the generation of models to support requirements elicitation, analysis, and design, to the synthesis of feature models in a product-line engineering context, to the generation of high-level models of early requirements [16]. For more details, we refer the reader to a recent comprehensive review [11]. In the following, we focus on UML model generation, which is most pertinent to our study.



Elevator System

REQ1. When the user presses the "Up" button on a floor, the Elevator System shall prioritize servicing the requested floor, moving upwards if necessary, and open its doors upon arrival.

REQ2. When the user presses the "Down" button on a floor, the Elevator System shall prioritize servicing the requested floor, moving downwards if necessary, and open its doors upon arrival.

REQ3. When the user presses any floor button inside the elevator, the Elevator System shall prioritize servicing the selected floor, moving upwards or downwards as needed, and open its doors upon arrival.

REQ4. When the overload sensor detects excessive weight in the elevator cabin, the Elevator System shall prevent further entry, emit an audible alarm, and display an overload warning. It shall not move until the excess weight is reduced and remain in the "Overload" state until the weight is within the acceptable limit.

REQ5. When the user presses any floor button inside the elevator while the system is in the "Overload" state, the Elevator System shall ignore the button press until the overload condition is resolved.

User
Elevator System
Overload Sensor
Outside the Elevator
Press "Up" or "Down" button
alt
[Press "Up" button]
Prioritize servicing requested floor
Move upwards if necessary
[Press "Down" button]
Prioritize servicing requested floor
Move downwards if necessary
Open doors upon arrival
Inside the Elevator
Press any floor button
opt
[System in "Overload" state]
Ignore button Press
Prioritize servicing selected floor
Move upwards or downwards as needed
Open doors upon arrival
Overload Scenario
Excessive weight detected
Prevent further entry
Emit audible alarm
Display overload warning
Do not move until weight is reduced
Remain in "Overload" state
User
Elevator System
Overload Sensor

Fig. 1. Example requirements for an “elevator system” and the corresponding sequence diagram.

The RE community has extensively investigated UML *class diagram* generation [4], [9], typically used for representing domain models (i.e., high-level abstraction of domain entities and their relations). Generating *sequence diagrams* or other behavioral representations has also been investigated in RE to some extent. Kof [17] generates message sequence charts (MSCs) from scenario descriptions using a rule-based approach. Yue *et al.* [7] present aToucan, a tool that extracts sequence diagrams from use case specifications expressed in a constrained natural language. The tool relies on parsing/transformation rules and uses an intermediate representation to pass from the specifications to the final models. More recently, Jahan *et al.* [8] present a rule-based approach that, unlike the other works, can be applied to free text use case specifications.

Large Language Models in RE. LLMs are deep neural networks that use a transformer architecture and are pre-trained on large amounts of text via self-supervised learning to capture the statistical regularities and semantic knowledge in natural language [18]. Recent LLMs are *generative* models that produce human-like responses to NL queries (also referred to as *prompts*). Existing surveys acknowledge that the application of LLMs in RE is particularly scarce compared to other software engineering areas, such as testing, code generation, and program repair [19], [20]. LLMs have been investigated in several contexts in RE, e.g., summarization from legal texts [21] and requirements traceability [22]. More relevant to our work, Chen *et al.* [12] evaluate the potential of GPT-4 for generating goal models using the textual grammar for the Goal-oriented Requirement Language (GRL) based on NL descriptions of the problem context. Chen *et al.* [14] evaluate GPT-3.5 and GPT-4 for generating class diagrams from NL descriptions. Camára *et al.* [13] conduct an exploratory study on using ChatGPT for class diagram generation. The authors conclude that iterations are needed to produce models of sufficient quality. Other studies in RE propose pattern catalogs of prompts for specific problems, such as classification and traceability [23], [24].

Research Gap. Most of the studies on model generation use rule-based approaches [11]. Such approaches require defining a complex set of heuristic rules, which are hardly maintainable and poorly adaptable. Saini *et al.*'s approach for class diagram generation is an exception [9] in that it combines rules with machine learning. However, their approach does not exploit LLM capabilities. Current works on model generation using LLMs [12]–[14] do not focus on sequence diagrams and they mainly use toy requirements instead of real-world specifications. In contrast to existing work, our study is, to the best of our knowledge, the first to: (1) target UML sequence diagram generation using LLMs; (2) consider industrial requirements specifications as input, belonging to different domains and having different formats.

III. Research Design

The overarching goal of our study is to examine the capability of ChatGPT to generate sequence diagrams. While sequence diagrams can serve various purposes, including code generation [25], our study specifically analyzes their role in *complementing* requirements, aiming to facilitate communication with stakeholders. Our study is guided by the following research questions (RQs):

RQ1: *What is the quality of the sequence diagrams generated from NL requirements by ChatGPT?* RQ1 aims to provide a *quantitative* evaluation of the quality degree of the diagrams generated by ChatGPT, thus giving an indication of its applicability in practical settings.

RQ2: *What are the issues emerging from using ChatGPT for generating sequence diagrams from NL requirements?* RQ2 aims to *qualitatively* explore the problem domain and produce a catalog of typical issues (e.g., incompleteness, low level of understandability) that can emerge when generating sequence diagrams using ChatGPT. By *issue*, we intend any observable problem in the generated diagram, possibly associated with some specific characteristic of the input requirements or limitations of ChatGPT.

We perform our study exclusively on ChatGPT based on the GPT3.5 model available through the web application[1]. Our rationale behind selecting GPT3.5 is that is free to use and provides an intuitive interface, which can be used by analysts who do not have the coding skills required to use the OpenAI API.

A. Data Collection

Datasets. Our data collection aimed at manually examining the sequence diagrams generated by ChatGPT and identifying issues that affect the quality of these diagrams. To achieve our goal, we collected 28 industrial requirements documents covering diverse application domains. The documents originate from three sources: (i) the "Ten Lockheed Martin Cyber-Physical Challenges"[2] containing ten requirements documents from the cyber-physical domain. (ii) The PURE dataset [26], containing 79 documents that cover multiple application domains and requirements formats (e.g., "shall" requirements, use case specifications). (ii) A dataset of user stories [27].

The criteria for selecting the documents are:
(i) *Diversity in structure, domain, and requirements types:* The documents cover diverse domains (18 in total, cf. Table I), such as healthcare, security, and railway, as well as different requirements types, namely "shall" type (s in Table I), user stories (us), and use case specifications (uc).
(ii) *Representativeness of real software projects:* Unlike toy requirements, the selected documents contain requirements from real industry projects.

Variants. From each of the 28 selected documents, two authors of this paper (A1 and A2) extracted one requirements subset that was considered amenable to be represented as a sequence diagram, and generated a model from it. In addition, A1 and A2 explicitly challenged ChatGPT by manually introducing a set of variants for each requirements subset. These variants encompass the intentional introduction of specific changes in the requirements, such as modifying, adding, or deleting a requirement. Additionally, the variants could introduce smells expected to occur in the requirements, e.g., ambiguity, inconsistency, and incompleteness. Introducing such changes exposes the LLMs to practical challenges in the RE field, i.e., the evolution of requirements throughout the project lifetime and the presence of smells. We selected only functional requirements, as sequence diagrams are more appropriate to

[1] https://chat.openai.com/

[2] https://github.com/hbourbouh/lm_challenges

represent behavior rather than quality aspects. A1 and A2—who have 7 to 13 years experience in quality requirements and NLP for RE, and practical confidence with sequence diagrams—incrementally defined the variants in a *greedy-like* manner, according to their intuition of what could be a realistic modification. A systematic generation of variants (i.e., introducing all possible changes in a given document) would have allowed a more complete exploration of the problem space. However, it was considered hardly feasible and would have constrained the creativity of the authors in introducing realistic changes.

Information about the requirements files and number of requirements in each requirements subset (REQ) is reported in Table I. A1 considered 44 variants of 19 subsets (the term "variant" also includes the original subset), and A2 considered 43 variants of 12 subsets—3 were common, and initially used to ensure alignment in the evaluation strategy. Our data collection resulted in analyzing 17 subsets of "shall" requirements (57 variants), 7 use case specifications (17 variants), and 4 user story documents (13 variants). In total, 87 variants were produced. These different numbers reflect the order of evaluation (i.e., "shall" requirements were evaluated first), as A1 and A2 incrementally selected the requirements documents and interrupted their evaluation when no additional issues emerged, i.e., a form of saturation [28] was reached. We acknowledge a bias due to this order, which could not be entirely mitigated.

Diagram Generation. We generated the sequence diagram for each variant by prompting ChatGPT using the *visualization generator pattern* [29]: "Generate a sequence diagram from these requirements so that I can provide it to Planttext to visualize it. Requirements: {list of requirements}". We selected Planttext[3] to visualize the sequence diagrams as it is web-based, free- and easy-to-use, and it applies the PlantUML textual language [4], a widely-used and human-readable language. In our analysis, we used solely the diagrams resulting from this prompt (i.e., no iterations were performed, and we used a separate session for each prompt). While sometimes we further prompted ChatGPT for corrections, the diagrams resulting from these iterations are not evaluated in this study.

Evaluation. For each generated diagram, A1 and A2 independently performed a critical evaluation, which was documented in a textual log file. The evaluation was performed according to the following quality criteria.

- **Completeness:** The diagram covers the content of all the requirements (external completeness [30]) with a sufficient degree of detail to communicate with potential stakeholders.
- **Correctness:** The diagram specifies a behavior that is coherent and consistent with the requirements.
- **Adherence to the standard:** The diagram is syntactically correct (i.e., it can be interpreted by PlanText[5]) and semantically sound (i.e., it uses constructs appropriately).

TABLE I
DATA COLLECTION RESULTS

File	Domain	REQ[†]	VAR[†]	ANN[†]
Triplex (s)	Cyber-physical System	8	13	Both
Inventory (s)	Inventory System	22	3	A2
Autopilot (s)	Cyber-physical System	14	9	Both
qheadache (s)	Gaming	11	5	Both
CentralTradingSys (uc)	E-commerce	1(5)[‡]	1	A2
wrac III (s)	Archiving	6	3	A2
datahub (us)	Data Management	67	3	A2
g02-uc-cm-req (uc)	Healthcare	1(11)	1	A2
g04-uc-req (uc)	Traffic Control	1(8)	3	A2
g05-uc-req (uc)	Football Digital System	5(37)	2	A2
pacemaker (s)	Healthcare	289	2	A2
UHOPE (us)	Healthcare	12	5	A2
FiniteStateMachine (s)	Cyber-physical System	13	1	A1
TustinIntegrator (s)	Cyber-physical System	4	1	A1
Regulators (s)	Cyber-physical System	10	1	A1
NonlinearGuidance (s)	Cyber-physical System	7	1	A1
NeuralNetwork (s)	Cyber-physical System	4	1	A1
EffectorBlender (s)	Cyber-physical System	5	1	A1
Euler (s)	Cyber-physical System	8	1	A1
caiso (s)	Black Start Generation	6	2	A1
eirene (s)	Railway	8	3	A1
ertms (s)	Railway	6	6	A1
evla-back (s)	Astronomy	8	1	A1
g04-recycling (us)	Recycling System	51	3	A1
g12-camperplus (us)	Camping System	13	2	A1
keepass (uc)	Security	1(11)	3	A1
peering (uc)	Networking	1(5)	2	A1
pnnl (uc)	Energy Diagnostics	1(11)	5	A1

[†] REQ: the number of analyzed requirements, VAR: the number of generated variants, ANN: the annotator who did the analysis.

[‡] Use Case (Steps): Note that we provide the number of use case specifications considered in the analysis as well as the total number of steps (between parentheses).

- **Degree of understandability:** The diagram is sufficiently clear, given the complexity of the requirements, and does not contain redundancies.
- **Terminological alignment:** The terminology used in the generated diagram aligns with the one used in the requirements.

The criteria were established by the authors through consensus, drawing from preliminary experiments oriented to identify relevant quality dimensions, and considering those criteria for requirements sets from ISO/IEC/IEEE 29148:2018(E) [31] that were considered applicable to models.

Each criterion was assessed according to a five-point ordinal scale where each integer indicates a degree of fulfillment of the criterion 1 = "Not fulfilled at all"; 2 = "Fulfilled to a minimal extent", 3 = "Partially fulfilled", 4 = "Mainly fulfilled" 5 = "Completely fulfilled". This information was then analyzed to answer **RQ1**. For each criterion, a textual justification for the score was provided, highlighting reflections on the observed issues. This information was then analyzed to answer **RQ2**.

More specifically, for each generated diagram, A1 and A2 included the following information in the evaluation log: (1) the change applied to the requirements, if any; (2) the evaluation score according to the above criteria, and the textual justification for the scores; (3) additional notes on the observed

[3] https://www.planttext.com/

[4] https://plantuml.com/guide

[5] PlantText may not be fully compliant with the UML standard, but we take its syntax as a reference to avoid manual checking of all the nuances of the reference standard.

issues; (4) a link to the conversation with ChatGPT or its textual copy. We make the logs and other related material available in our online annex [32].

In our evaluation, we did not use a manually defined ground truth for two reasons: (a) more than one diagram exists that satisfies the same requirements; (b) existing ground truths are limited (e.g., three diagrams in [8]), and here we wanted to have a wider perspective on possible issues. Additional reflections on the rationale of the study design are in Sect. V.

B. Data Analysis

For **RQ1**, we first assessed that A1 and A2 had similar interpretations of the score values according to the scale. To this end, an independent cross-evaluation was performed. A1 inspected and scored 15 of the diagrams produced by A2, and vice versa—a total of 30 models were cross-evaluated. The agreement between A1 and A2, computed through a square-weighted Cohen's Kappa [33], led to $\kappa = 0.67$, indicating substantial agreement. With square-weighted Kappa, disagreements are weighted according to their squared distance from perfect agreement, thus penalizing larger disagreements in the scale. We further used the nonparametric Wilcoxon signed rank test for each criterion to check whether the average scores significantly differed from the mean value of 3, as done e.g., in [34], thus suggesting a high degree of fulfillment of the criterion. Specifically, we tested the null hypothesis: *The scores for [criterion] do not differ from the mean value*, considering $\alpha = 0.05$. We also evaluated the effect size with Cohen's d [35]. For this evaluation, we used solely cases that did not include modifications or smells. The reason is that such alterations could lead to inaccurate diagrams, while here we want to check the reliability of ChatGPT starting from well-formed requirements.

For **RQ2**, an author of the paper not involved in the evaluation (A3) performed the thematic analysis according to Clarke and Braun [36] through semi-open coding in NVivo, using the logs produced during the data collection phase. The analysis aimed to identify and classify issues encountered during the generation of sequence diagrams. Closed codes are the criteria (completeness, correctness, etc.) that we regard as high-level (HL) codes (our pre-defined meta-level codes based on the five criteria discussed earlier). For each HL code, open coding was performed. A3 went through the logs and annotated them with low-level (LL) codes, asking for clarifications when some of the logs were not entirely understandable. For example, the statement "Failure management is also missing as a component" was coded as *Missing Component*, under the HL code *Completeness*. 135 LL codes were introduced at this stage, partitioned into different HL ones, plus an additional one, *General Issues*, including LL codes that could not be directly associated with the evaluation criteria. Then, the LL codes were revised by A1, who also inspected the associated logs and suggested removing or merging some of the codes. This sanitization process resulted in 62 LL codes, which were then aggregated into 23 mid-level (ML) codes—still grouped according to the HL codes, plus *General Issues*. For instance, *Missing Component* and *Missing Condition* were grouped under the ML code *Summarization Issues*, under the HL code *Completeness*. A2 finally inspected the results of the thematic analysis. The involvement of three subjects was aimed at mitigating the inherent subjectivity of thematic analysis. In defining the codes, we did not differentiate among requirements formats. Furthermore, we did not systematically trace requirements modifications with issues. The analysis of format-related aspects and the correlation between issues and modifications requires a more systematic analysis, which is outside the scope of this study. Here, we are mainly interested in general common issues that can inspire research questions for future investigations.

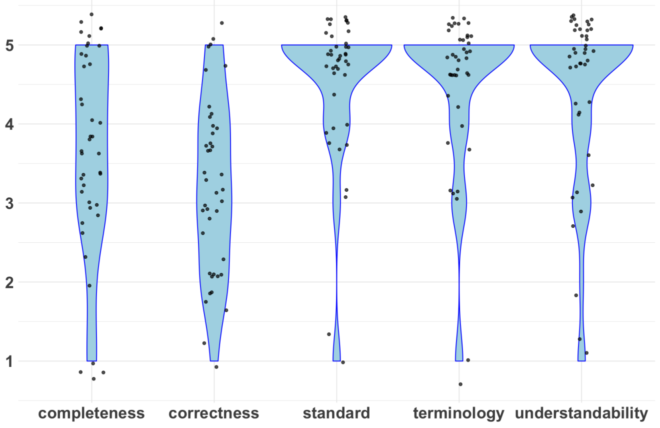


Fig. 2. Violin plots for the different evaluation criteria.

IV. Execution and Results

A. RQ1: Quality of the Diagrams

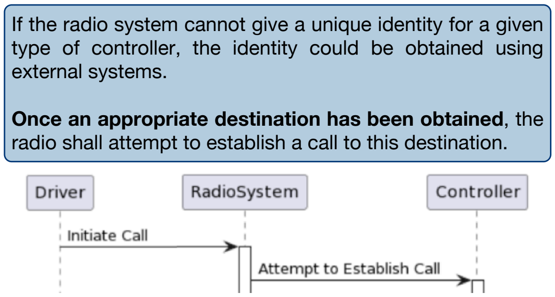
Fig. 2 shows the violin plots with jittered points resulting from evaluating the different criteria on the generated models, considering well-formed requirements as input. Table II reports the statistics and the results of the tests. We see that completeness, understandability, adherence to the standard, and terminological alignment are significantly above the mean value (p-value $\leq$ 0.05) with medium (d $>$ 0.2) to large (d $>$ 0.8) effect size—reference values from Cohen [35]. This indicates a sufficient degree of model quality for these criteria. Instead, scores for correctness issues do not significantly differ from the mean (p-value $>$ 0.05), meaning that, on average, the criterion was not adequately fulfilled. In addition, by looking at the violin plot, the distribution of completeness, although skewed towards high values, is still not optimal, with several 3 and 4 scores. These results suggest that a requirements analyst who wants to generate diagrams with ChatGPT needs to be careful and check its output's correctness and completeness, paying attention to the issues we outline in the following section.

B. RQ2: Emerging Issues

In the following, we discuss HL and ML codes emerging from the thematic analysis (see [32] for LL codes), also

TABLE II
STATISTICS AND TEST RESULTS

Criterion	Mean	p-value	Cohen d	Eff. S.
complet.	3.634146	0.003652	0.503258	Med.
correctness	3.219512	0.101543	0.194357	Small
standard	4.536585	1.4E-07	1.572471	Large
understand.	4.365854	6.35E-07	1.22766	Large
terminology	4.487805	2.13E-07	1.447751	Large



If the radio system cannot give a unique identity for a given type of controller, the identity could be obtained using external systems.

Once an appropriate destination has been obtained, the radio shall attempt to establish a call to this destination.

Fig. 3. "Summarization Issues".

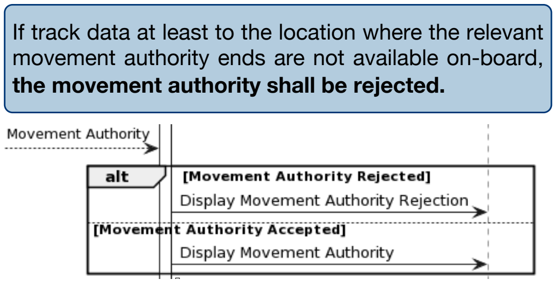
presenting evidence in the form of requirements and diagram excerpts. Please note that each example is a *portion* of a requirements group and a *zoom* of the associated diagram. Complete artifacts are in our evaluation logs [32].

1. Completeness.

Summarization Issues: Observations highlight instances where the generated content tends to be excessively summarized. Missing elements encompass requirements or fragments thereof, components, function calls, conditions, and messages. While sequence diagrams unavoidably summarize the content of the requirements, as they are meant to complement and not replace them, it is important to ensure that all the *relevant* information is present. It appears that ChatGPT finds it hard to distinguish between relevant and inconsequential information. An example is represented in Fig. 3, concerning a radio system for a train. We see that the first requirement and the statement "Once an appropriate destination has been obtained" are ignored, and the Driver initiates the call, assuming that the destination (i.e., the Controller's identity) was already obtained.

Poor Requirements Quality and Model Omissions: One interesting phenomenon is the possible effect of requirements quality on the omission of relevant information. It appears that when requirements are hard to understand or ambiguous, the information associated with them is somewhat "hidden" by abstracting their details, which leads to incomplete diagrams. In Fig. 4, concerning the requirements for a train control system, we see that a condition that is hardly understandable ("If track data at least to the location where the relevant movement authority ends are not available on-board") is basically concealed behind the alternative "Movement Authority Rejected".

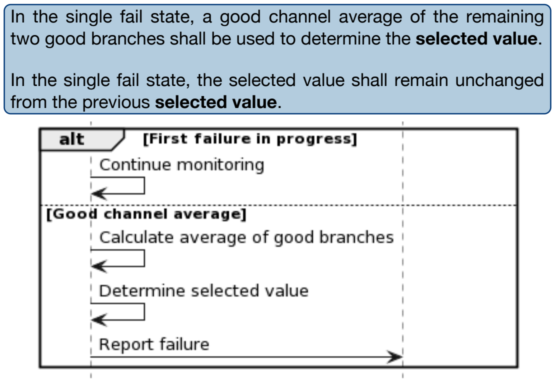
Inconsistency and Model Omissions: Similar to the previous code, inconsistency between requirements can be associated with situations in which ChatGPT hides the conflict, thus leading to omissions. Fig. 5 considers a failure management system with triple redundancy. We have two conflicting requirements—different selected values for the same state. The conflict is hidden by the function call "Determine selected value", which does not specify how the value should be determined.



If track data at least to the location where the relevant movement authority ends are not available on-board, **the movement authority shall be rejected.**

Movement Authority
alt
[Movement Authority Rejected]
Display Movement Authority Rejection
[Movement Authority Accepted]
Display Movement Authority

Fig. 4. "Poor Requirements Quality and Model Omissions".

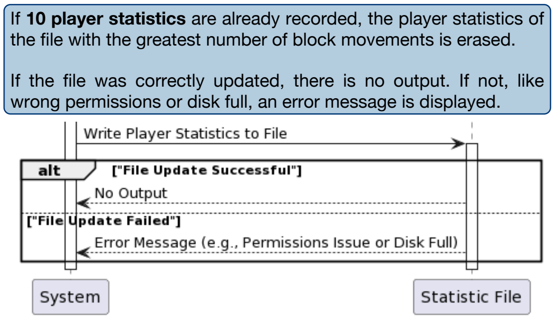


In the single fail state, a good channel average of the remaining two good branches shall be used to determine the **selected value**.

In the single fail state, the selected value shall remain unchanged from the previous **selected value**.

Fig. 5. "Inconsistency and Model Omissions", "Inconsistency and Model Incorrectness", and "Incoherence Manifestations".

Poor Precision in Timing and Numbers: As LLMs primarily focus on language, ChatGPT notably encounters difficulties with numbers, timing, and mathematical thinking in general. This challenge becomes evident when the requirements involve numerical constraints, as in Fig. 6, where the requirements of a computer game are considered. The requirement with the numerical condition ("If 10 player statistics are already recorded") is entirely ignored.



If **10 player statistics** are already recorded, the player statistics of the file with the greatest number of block movements is erased.

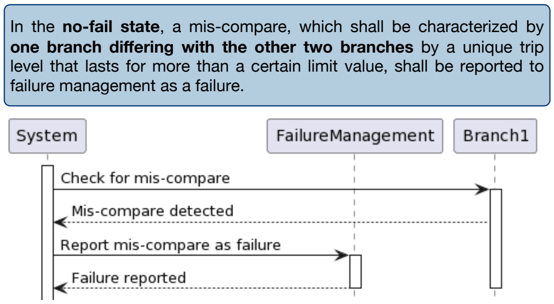
If the file was correctly updated, there is no output. If not, like wrong permissions or disk full, an error message is displayed.

Write Player Statistics to File
alt
["File Update Successful"]
No Output
["File Update Failed"]
Error Message (e.g., Permissions Issue or Disk Full)
System
Statistic File

Fig. 6. "Poor Precision in Timing and Numbers".

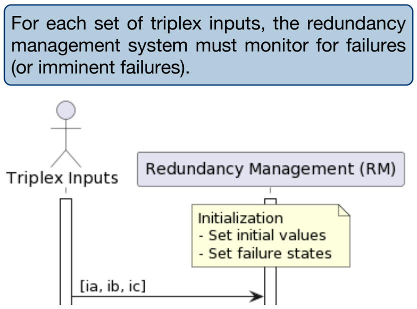
2. Correctness.

Incorrect Interactions: The models can include an incorrect order of function calls/messages or an incorrect flow of actions with respect to what is expressed in the requirement. This is the case of Fig. 7, again considering a failure management



In the **no-fail state**, a mis-compare, which shall be characterized by **one branch differing with the other two branches** by a unique trip level that lasts for more than a certain limit value, shall be reported to failure management as a failure.

Fig. 7. “Incorrect Structure” and “Incorrect Interaction”.



For each set of triplex inputs, the redundancy management system must monitor for failures (or imminent failures).

Fig. 8. “Incorrect Component/Actor”.

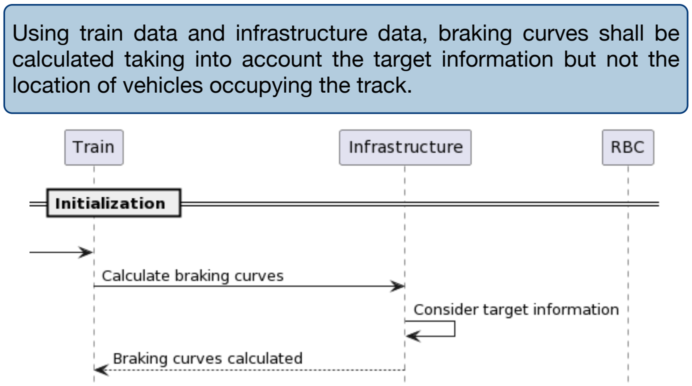
system with triple redundancy. Here, only one branch is considered to detect a mis-compare, instead of three branches, as the requirement indicates (“one branch differing from the other two branches”).

Incorrect Component/Actor: The models include components or actors that should be messages or functions. Fig. 8 exemplifies this case, where “Triplex input” is treated as an actor, while it should be a data structure passed through a message from the sensors.

Incorrect Structure: ChatGPT is sometimes not sufficiently effective in capturing conceptual abstract elements, such as states and control loops, which help to correctly structure the models. In other cases, function calls are not appropriately grouped and are scattered throughout the diagram. In Fig. 7, the “no-fail state” is incorrectly overlooked (this can also be regarded as a summarization issue), and the diagram misses a control loop, which is needed as all the actions are performed cyclically.

Unclear Requirements and Model Incorrectness: When ChatGPT encounters vague or ambiguous aspects in the input requirements, it may produce content open to interpretation, potentially leading to inaccuracies. The requirement for a train control system in Fig. 9 uses passive voice (“braking curves shall be calculated”), which leads ChatGPT to think that the infrastructure calculates the braking curve of the train, although the train system normally computes it.

Inconsistency and Model Incorrectness: ChatGPT may struggle to produce accurate representations when encountering inconsistencies or conflicting information. In Fig. 5, the inconsistency between the requirements is not only hidden but appears to produce incorrect behaviour. When the diagram condition “First failure in progress” is true (i.e., “in the single fail state”, in the requirements’ terminology), only the second requirement is considered (i.e., “the value shall remain unchanged”), while the first is not.

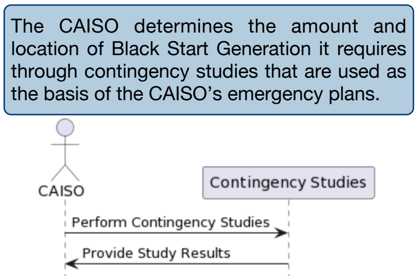


Using train data and infrastructure data, braking curves shall be calculated taking into account the target information but not the location of vehicles occupying the track.

Fig. 9. “Unclear Requirements and Model Incorrectness”.

3. Adherence to the Standard.

Syntax Errors: In some cases, the output cannot be interpreted by PlanText, raising syntactic errors. Even after pinpointing the specific error, ChatGPT cannot recover from it, forcing the user to modify the diagram manually.



The CAISO determines the amount and location of Black Start Generation it requires through contingency studies that are used as the basis of the CAISO’s emergency plans.

Fig. 10. “Semantic Errors” and “Additional Terms”.

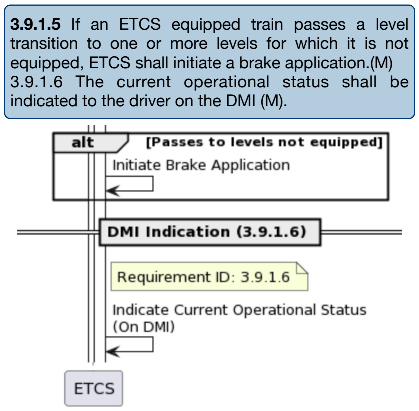
Semantic Errors: The usage of constructs is sometimes semantically incorrect. A case is the usage of function calls instead of messages, as shown by the call to “Provide study results” in Fig. 10, concerning a system that should have been a return message.

4. Terminology.

Additional Terms: Novel terms that were not originally present in the requirements are introduced by ChatGPT, which hinders the coherence of the models with respect to the requirements. This is shown in Fig. 10, about a standard for energy grids, where “Study Results” are never mentioned.

Relevant Terms Missing: Terms that were considered relevant during the analysis are not reported in the model. Again, Fig. 10 shows that the term “Black Start Generation”, which appears to be relevant for the requirement, as expressed in capital letters, is missing from the diagram.

Inconsistent Terminology: The terms introduced are not consistent with the original ones, although they appear to express a similar meaning. For example, in Fig. 5, the expression “In the



3.9.1.5 If an ETCS equipped train passes a level transition to one or more levels for which it is not equipped, ETCS shall initiate a brake application.(M)
3.9.1.6 The current operational status shall be indicated to the driver on the DMI (M).

alt
[Passes to levels not equipped]
Initiate Brake Application
DMI Indication (3.9.1.6)
Requirement ID: 3.9.1.6
Indicate Current Operational Status (On DMI)
ETCS

Fig. 11. "Traceability Challenges".

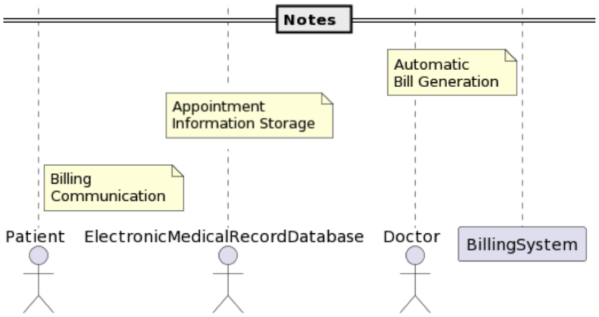

Notes
Automatic Bill Generation
Appointment Information Storage
Billing Communication
Patient
ElectronicMedicalRecordDatabase
Doctor
BillingSystem

Fig. 12. "Misplaced Information".

single fail state" appears to be replaced by the condition "First failure in progress". It should be, however, acknowledged that terminological issues are often acceptable if they preserve the intended meaning of the original terms.

5. Understandability. Due to the complexity of models with understandability issues, here we report only codes that can be represented synthetically through diagram excerpts.

Traceability Challenges: Assessing completeness and correctness can be hard without clearly tracing the requirements and the diagram. Tracing information is sometimes included as notes, which, although helpful in principle, can be incomplete or inaccurate. Fig. 11 shows a case in which only one requirement is traced while the other is ignored, although the associated action is correctly displayed ("Initiate Brake Application").

Incoherence Manifestations: When requirements are inconsistent, unclear, or incomplete, ChatGPT attempts to produce a model, but often at the cost of understandability, e.g., in Fig 5, it is unclear what the condition "Good channel average" means.

Misplaced Information: Sometimes, clarifying notes are added, which can be useful to facilitate understandability (cf. Fig. 8). However, they are occasionally not placed close to the model parts they refer to, as in Fig. 12, where notes are aggregated in the bottom part of the diagram.

Presence of Redundancy: ChatGPT can introduce redundant or superfluous information, which, although it does not impact the correctness of the model, affects its understandability.

Actor Overrepresentation: Sometimes too many actors are introduced, which makes the sequence diagram too wide and hard to navigate.

Too Detailed: The diagrams should faithfully represent the requirements, but sometimes the generated diagrams are too detailed with nested conditions and loops to be easily navigated. In such cases, a higher degree of abstraction would be expected.

6. General Issues.

Memory-Induced Hallucinations: Occasionally, ChatGPT appears to hallucinate, i.e., it provides an output inconsistent with the query. We observed that this is likely due to interactions with the user. Specifically, we sometimes observed that requirements used in previous independent sessions and generated models appeared to influence subsequent sessions.

Ignored Requirements Modifications: Possibly due to ChatGPT's memory of previous chat sessions, modifications to the requirements are sometimes ignored. This can complicate an iterative process in which requirements are incrementally introduced, and models are generated through multiple iterations, as in the scenario illustrated by [3].

Lack of Contextual Understanding: ChatGPT might not have domain-specific knowledge or context, which can be crucial for accurately translating requirements into sequence diagrams. Given the token limit of ChatGPT, it might not be easy to provide the entire context required for ChatGPT to represent a sequence diagram accurately. For instance, in some cases where requirements had cross-references to other parts, the generated output exhibited lower quality, possibly due to the lack of context. Another case is Fig. 7, where ChatGPT appears to miss the technical knowledge of how a triple redundancy failure management system should normally work.

Variability of the Output: Given the same query and a set of requirements, the output of ChatGPT can largely vary. Although this is a well-known issue [13], it should be noted that different diagram representations can be appropriate for the same requirements, and the generation of alternatives can even allow the user to choose a preferred one. Variability in multiple runs is not a substantial problem *per-se*, but rather an asset, if the correctness of the output is ensured.

V. Discussion

In the following, we discuss our findings, and speculate on possible solutions to the identified problems.

Requirements Quality and Diagram Quality. Correctness of the diagrams is the main issue observed, followed by completeness. We noticed that these aspects could be associated with requirements quality issues, as ChatGPT could neglect or conceal requirements content that cannot be interpreted unequivocally. To address this issue, one should consider performing quality checks on the requirements, either through manual inspection, automatic tools [16], or through the support of ChatGPT itself, as done in recent works [10], [37]. The relation between requirements defects and diagram quality should be confirmed by more rigorous experiments stemming from our exploratory study. If this relation is confirmed, diagram generation with ChatGPT could help spot requirements quality issues. ChatGPT could play the role of a fictional conversational partner that strives to comprehend the content of the requirements and then demonstrates its understanding through the creation of diagrams. The generation of a low-quality diagram may suggest that further improvement of the requirements is needed.

Incremental/Interactive Diagram Definition. An RE process in which ChatGPT acts as an assistant does not rule out the primary role of an experienced requirements analyst. Multiple prompt iterations can be needed before ChatGPT can produce a diagram in the desired form and with the expected quality. These iterations should help ChatGPT self-correct its output and address more complex real-world issues. In practical contexts, requirements may need to be first decomposed by functionality and then transformed into clear lists of steps amenable to a sequential representation. Furthermore, other types of diagrams (e.g., domain models in the form of class diagrams and goal models) may be required to create higher-level abstractions or different views. ChatGPT can provide support in this regard [10], [12], [13], [38]. In other contexts, one may start from early requirements and incrementally use ChatGPT to produce diagrams that help to refine the requirements further [3]. Incremental prompt engineering strategies are needed to accommodate these diverse RE contexts and identify the best way to exploit the synergy between requirements analysts and LLMs.

The Role of Domain/Contextual Knowledge. We have observed that requirements that included term definitions and some additional context allowed ChatGPT to provide richer and more accurate representations. While ChatGPT has been trained on large amounts of documents, it may lack the technical knowledge often required to comprehend the requirements. Within the token limits, partial context and knowledge can be provided in a prompt. When one uses the ChatGPT API, fine-tuning with context-specific data is also possible. Furthermore, probing prompts can be used first to verify whether the LLM correctly understands a requirement (e.g., asking to rephrase it). When its understanding appears to be incorrect or only partial, the user can provide the additional domain knowledge that the LLM lacks.

Improving Understandability. Understandability of diagrams is generally good. However, an apparently clear diagram might lead an analyst to believe it is correct, even if it is not. ChatGPT spontaneously introduced notes and traceability information, which—although sometimes imperfect—can greatly help the interpretation of diagrams. Explicitly requesting these explanatory elements could provide better transparency, aiding users in comprehending the rationale behind certain decisions or representations, and assessing correctness. ChatGPT is also notoriously verbose in its answers, but having NL explanations—besides diagrams with notes—and explicitly asking for them can provide useful information. This can be used as additional documentation for the diagrams when these are exchanged with stakeholders.

Empirical Research with ChatGPT. We acknowledge that the research design adopted in this study is unorthodox. However, the study matter is novel, and appropriate validation methods for LLMs are still under development [39]. We believe that our work contributes with a research design that can be adopted by other exploratory studies using LLMs. Following the empirical research framework by Stol and Fitzgerald [40], our research is positioned between a judgment study (where subject matter experts express their opinions) and a sample study (where objects/subjects are sampled from a population and analyzed/surveyed). These types of studies aim for generalizability of the findings over different contexts but cannot inherently control the behavior of subjects/objects and cannot achieve context specificity, as, e.g., in case studies. We deem this design appropriate for situations in which one cannot fully control a phenomenon, such as ChatGPT's behavior, and wants to perform an exploratory investigation over different realistic situations, i.e., different requirements sample cases.

VI. Threats to Validity

Construct Validity. This study involved a quantitative assessment of ChatGPT's reliability based on five criteria, evaluated using a scoring scale. The use of a scoring scale introduces a potential subjectivity threat in criteria interpretation and evaluation. To address this, we established a shared definition of the criteria, based on a selection of model-relevant criteria inspired by [31], and preliminary experiments. More criteria can be added in future works. We also computed a Cohen's kappa on a subset of the models, indicating substantial agreement, which mitigates the subjectivity threat.

Internal Validity. A1 and A2 did not use a gold standard, so they used their judgment to assess that the models were actually faithful to the requirements. This subjectivity threat cannot be entirely mitigated, but it is arguably justified by the exploratory nature of the study, and mitigated by the experience of the assessors in RE. Furthermore, the evaluation of A1 and A2 was documented through NL logs, a process inherently subjective. To enhance objectivity, the thematic analysis was conducted by A3, who was not part of the initial evaluation, providing an impartial perspective. To ensure coherence, A1 and A2 contributed to the sanitization and consolidation of themes, forming a triangulation approach that partially mitigates subjectivity threats. The ChatGPT memory of previous sessions (cf. Sect. IV) could have also affected the results. However, this is also the behavior that one should expect in practice, and thus makes our evaluation more realistic.

External Validity. Our results stem from an exploration of the ability of ChatGPT to generate sequence diagrams from requirements. The exploration is nonsystematic and unavoidably incomplete. The authors performed their analyses independently, which mitigates the threat of an incomplete exploration, as they introduced different variants in the original models, and considered different documents. The selected documents could have biased the evaluation. However, we considered different requirement types in our exploration, which extends the scope of validity of our conclusions, and their generalizability.

VII. Conclusion

In this study, we explored the reliability of ChatGPT in generating UML sequence diagrams from NL requirements. The evaluation revealed promising results in terms of terminological consistency, understandability, and adherence to the standard of the generated models. However, challenges emerged in terms of model *completeness* and *correctness*, particularly when dealing with ambiguous or inconsistent requirements. We also observed occasional hallucinations, and, in some instances, the model demonstrated limitations in contextual understanding and appeared to lack domain-specific knowledge. These findings have important implications for leveraging LLMs in RE. Providing additional contextual information and more domain knowledge can be expected to improve ChatGPT's model generation capability. Furthermore, devising incremental prompting strategies with the human-in-the-loop to decompose, verify, and refine the requirements, can help to enhance the correctness of the produced models.

References

[1] R. Jolak, M. Savary-Leblanc, M. Dalibor, A. Wortmann, R. Hebig, J. Vincur, I. Polasek, X. Le Pallec, S. Gérard, and M. R. Chaudron, "Software engineering whispers: The effect of textual vs. graphical software design descriptions on software design communication," *Empirical software engineering*, vol. 25, pp. 4427–4471, 2020.

[2] S. Wagner, D. M. Fernández, M. Felderer, A. Vetrò, M. Kalinowski, R. Wieringa, D. Pfahl, T. Conte, M.-T. Christiansson, D. Greer *et al.*, "Status quo in requirements engineering: A theory and a global family of surveys," *ACM TOSEM*, vol. 28, no. 2, pp. 1–48, 2019.

[3] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with circe," *ASE*, vol. 13, pp. 107–167, 2006.

[4] C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, "An active learning approach for improving the accuracy of automated domain model extraction," *ACM TOSEM*, vol. 28, no. 1, pp. 1–34, 2019.

[5] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: an extended systematic mapping study," *REJ*, vol. 24, pp. 133–160, 2019.

[6] "Unified modeling language (UML) 2.5.1 core specification," https://www.omg.org/spec/UML, 2017.

[7] T. Yue, L. C. Briand, and Y. Labiche, "aToucan: an automated framework to derive uml analysis models from use case models," *ACM TOSEM*, vol. 24, no. 3, pp. 1–52, 2015.

[8] M. Jahan, Z. S. H. Abad, and B. Far, "Generating sequence diagram from natural language requirements," in *REW'21*. IEEE, 2021, pp. 39–48.

[9] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Automated, interactive, and traceable domain modelling empowered by artificial intelligence," *SoSym*, pp. 1–31, 2022.

[10] C. Arora, J. Grundy, and M. Abdelrazek, "Advancing requirements engineering through generative ai: Assessing the role of llms," *arXiv preprint arXiv:2310.13976*, 2023.

[11] S. Ahmed, A. Ahmed, and N. U. Eisty, "Automatic transformation of natural to unified modeling language: A systematic review," in *SERA'22*. IEEE, 2022, pp. 112–119.

[12] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, and D. Varró, "On the use of gpt-4 for creating goal models: an exploratory study," in *REW'23*. IEEE, 2023, pp. 262–271.

[13] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml," *SoSym*, pp. 1–13, 2023.

[14] K. Chen, Y. Yang, B. Chen, J. A. H. López, G. Mussbacher, and D. Varró, "Automated domain modeling with large language models: A comparative study," in *MODELS'23*. IEEE, 2023, pp. 162–172.

[15] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied thematic analysis*. sage publications, 2011.

[16] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," *CSUR*, vol. 54, no. 3, pp. 1–41, 2021.

[17] L. Kof, "Scenarios: Identifying missing objects and actions by means of computational linguistics," in *RE'07*. IEEE, 2007, pp. 121–130.

[18] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *CSUR*, vol. 56, no. 2, pp. 1–40, 2023.

[19] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[20] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *arXiv preprint arXiv:2308.10620*, 2023.

[21] C. Jain, P. R. Anish, A. Singh, and S. Ghaisas, "A transformer-based approach for abstractive summarization of requirements from obligations in software engineering contracts," in *RE'23*. IEEE, 2023, pp. 169–179.

[22] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *REW'23*. IEEE, 2023, pp. 455–464.

[23] K. Ronanki, B. Cabrero-Daniel, J. Horkoff, and C. Berger, "Requirements engineering using generative ai: Prompts and prompting patterns," *arXiv preprint arXiv:2311.03832*, 2023.

[24] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," *arXiv preprint arXiv:2303.07839*, 2023.

[25] D. Kundu, D. Samanta, and R. Mall, "Automatic code generation from unified modelling language sequence diagrams," *IET Software*, vol. 7, no. 1, pp. 12–28, 2013.

[26] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *RE'17*. IEEE, 2017, pp. 502–505.

[27] F. Dalpiaz and A. Sturm, "Conceptualizing requirements using user stories and use cases: a controlled experiment," in *REFSQ'20*. Springer, 2020, pp. 221–238.

[28] V. Braun and V. Clarke, "To saturate or not to saturate? questioning data saturation as a useful concept for thematic analysis and sample-size rationales," *Qualitative research in sport, exercise and health*, vol. 13, no. 2, pp. 201–216, 2021.

[29] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatGPT," *arXiv:2302.11382*, 2023.

[30] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," *IST*, vol. 45, no. 14, pp. 993–1009, 2003.

[31] "Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering," *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104, 2018.

[32] F. Alessio, A. Sallam, and A. Chetan, "Model Generation from Requirements with LLMs: an Exploratory Study - Replication Package," Apr. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10579731

[33] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, 1960.

[34] S. Abrahão, E. Insfran, J. A. Carsí, and M. Genero, "Evaluating requirements modeling methods based on user perceptions: A family of experiments," *Inf. Sci.*, vol. 181, no. 16, pp. 3356–3378, 2011.

[35] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[36] V. Clarke and V. Braun, "Thematic analysis," *The journal of positive psychology*, vol. 12, no. 3, pp. 297–298, 2017.

[37] J. Zhang, Y. Chen, N. Niu, and C. Liu, "A preliminary evaluation of ChatGPT in requirements information retrieval," *arXiv:2304.12562*, 2023.

[38] K. Ronanki, C. Berger, and J. Horkoff, "Investigating chatgpt's potential to assist in requirements elicitation processes," in *SEAA'23*. IEEE, 2023, pp. 354–361.

[39] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using LLMs in software engineering," *arXiv:2312.08055*, 2023.

[40] K.-J. Stol and B. Fitzgerald, "Guidelines for conducting software engineering research," in *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 27–62.