IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 August 2024, accepted 17 September 2024, date of publication 19 September 2024,
date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3464242

RESEARCH ARTICLE

Can GPT-4 Aid in Detecting Ambiguities, Inconsistencies, and Incompleteness in Requirements Analysis? A Comprehensive Case Study

TASLIM MAHBUB, (Member, IEEE), DANA DGHAYM, AADHITH SHANKARNARAYANAN, (Member, IEEE), TAUFIQ SYED, SALSABEEL SHAPSOUGH, AND IMRAN ZUALKERNAN, (Member, IEEE)
Department of Computer Science and Engineering, American University of Sharjah, Sharjah, United Arab Emirates
Corresponding author: Taslim Mahbub (b00075270@aus.edu)

This work was supported in part by the Open Access Program from American University of Sharjah.

ABSTRACT Effective software projects hinge on robust requirements, yet flawed requirements often lead to costly delays and revisions. While tools have been developed to identify defects in Software Requirements Specifications (SRS), the advent of Large Language Models (LLMs) like GPT-4 presents new opportunities for enhancing requirements quality. However, the potential of LLMs in this realm remains largely unexplored, particularly in the context of large-scale industrial documents. To bridge this gap, we investigate the efficacy of zero-shot GPT-4 in various requirements analysis tasks using an industrial software specification document. Our study evaluates LLM performance in detecting defects, such as ambiguities, inconsistencies, and incompleteness, while also analyzing GPT-4's ability to identify issues across version iterations and support technical experts in requirements analysis. Qualitatively, we identify key limitations of LLMs in defect detection, notably their inability to cross-reference throughout the document and their constrained understanding of specialized contexts. Quantitatively, we find that while LLMs excel in identifying incomplete requirements (precision 0.61), their performance is less impressive in detecting inconsistencies (precision 0.43) and ambiguities (precision 0.39). Although GPT-4 demonstrates promise in automating early defect detection across versions and providing accurate technical answers, our results underscore that they cannot entirely replace human analysts due to their lack of nuanced domain knowledge in a zero-shot setting. Nevertheless, avenues like few-shot learning and complex prompt design offer the potential to enhance LLM precision in defect detection.

INDEX TERMS Ambiguity, completeness, GPT, inconsistency, large language models (LLMs), requirements engineering, software engineering, software requirements specifications (SRS).

I. INTRODUCTION

The requirements document of any software development project plays a crucial role in influencing the project's success [1]. Misinterpretations or omissions in requirements analysis can lead to considerable discrepancies in the final product despite the deployment of ideal development efforts [2]. The Software Requirements Specification (SRS) document typically specifies the software's intended functionalities and capabilities in natural language [3], [4]. However, discrepancies in understanding this document among project team members can result in a final product that deviates from the client's expectations. Recognizing this, the early identification of errors within the SRS document

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

is a pivotal strategy for conserving financial resources and time during the project lifecycle [5], [6]. Consequently, the development and application of techniques that facilitate the accurate detection of such defects are of paramount importance. This paper explores the use of Large Language Models (LLMs) [7] in enhancing the quality and efficiency of software development projects.

Several specialized requirements analysis approaches are used to ensure that the software meets its intended objectives, and stakeholder needs [8], [9]. For example, abstract interpretation offers a method for approximating program behavior, which helps detect errors across all possible inputs [10]. Another approach is automated theorem proving that employs computational methods to automatically verify logical statements or properties of software systems [11]. Additionally, lightweight formal methods provide practical formal support for software development processes without the full rigor of traditional methods, offering techniques such as model checking and property-based testing [12]. Moreover, research focused on analyzing the SRS document is a crucial first step in confirming that requirements meet stakeholder expectations without prematurely investing time and resources in extensive technical implementation [1], [2]. Analyzing the SRS document helps identify and rectify defects early in the software development process.

Traditionally, Natural Language Processing (NLP) based solutions were implemented to detect language markers and indicate SRS document defects [1]. Contemporary studies increasingly demonstrate the superior potential of Large Language Model (LLM)-based approaches over rule-based NLP systems for detecting ambiguities within SRS documents [13]. For instance, research in [13] illustrates that GPT-3.5, a recent LLM, can effectively identify ambiguities, albeit within the confines of simpler case studies. Arora et al. [14] employed ChatGPT-based agents for requirements elicitation, revealing that these models could uncover useful requirements potentially overlooked by human experts. However, ChatGPT lacked the comprehensive accuracy that human analysts provide [14]. Finally, Luitel et al. [15] utilized a BERT-based LLM to augment specific words within a dataset, showcasing that LLMs could significantly enhance the completeness of requirement specifications. These developments suggest a shifting paradigm towards more sophisticated, AI-driven approaches in requirements engineering.

This work explores using LLMs to detect three significant defects in software requirements: ambiguity, inconsistency, and incompleteness. Using zero-shot learning with GPT-4, a comprehensive software requirements document for a mechanical lung ventilator (MLV) system [16] was analyzed to detect defects in real-world scenarios. In addition, we also (1) analyze the capabilities of GPT-4 in the early detection of future versioning issues and (2) evaluate GPT-4's ability to answer technical modeling questions for a given SRS, thereby aiding experts in requirements analysis. The primary contributions of this paper are as follows:

- A detailed examination of the performance of GPT-4 in requirements analysis tasks.
- An exploration of the precision and significance of errors identified by GPT-4 in an industrial SRS document across three defect types: ambiguity, inconsistency, and incompleteness.
- A comparative analysis of the defect-identification capabilities of human experts versus those of an LLM.
- Development of a versioning history to track defects recognized by the GPT-4 model that were also flagged by human experts modeling the system.
- Comprehensive assessments demonstrate that while GPT-4 exhibits potential in various requirements analysis tasks, they also possess inherent vulnerabilities that necessitate cautious application.

The paper is structured as follows: Section II provides an overview of requirements engineering and the role of LLMs. Section III outlines the methodologies and procedures employed along with details of the case study and objectives. The results of the GPT-4's ability to detect defects are presented in Section IV. This is followed by an additional experiment on early-versioning defect detection in Section V. The LLMs' capability to assist experts in requirements analysis is discussed in Section VII. An overall discussion is provided in Section in Section VIII. The paper concludes in Section IX, where we summarize our findings and outline directions for future research.

II. BACKGROUND

A. REQUIREMENTS ENGINEERING

Requirements engineering serves as a fundamental phase in the software development lifecycle, aimed at ensuring that the software ultimately meets the intended needs and constraints of stakeholders [1]. It unfolds in four stages [14]: Firstly, requirements elicitation distills user and system needs through domain exploration and stakeholder consultation. Secondly, specification transcribes these needs into detailed, structured documentation for clear guidance. Thirdly, analysis scrutinizes these requirements to resolve conflicts and refine priorities, ensuring alignment with project goals. Lastly, validation rigorously assesses the requirements to confirm their accuracy and completeness, ensuring they are fit for purpose. Requirements analysis aims to interpret, assess, and refine these NL requirements to ensure their clarity and thoroughness. The quality of requirements is significantly enhanced through this analysis.

Formal verification approaches are integral to requirements analysis, offering systematic techniques to ensure that system specifications meet desired properties and constraints [17]. These approaches employ mathematical modeling, logic, and automated reasoning tools to rigorously verify whether a system satisfies its requirements [18]. One common method is model checking [12], which exhaustively explores all possible states of a formal model to verify whether it meets specified properties or satisfies given requirements. Another novel method proposed uses an

ontology-driven method in which requirements are formulated with boilerplates containing placeholders filled with ontology elements [19].The ontology ensures a consistent semantic interpretation of the requirements, enabling automated semantic analyses to identify issues in the SRS.

Recent works in requirements analysis have begun to leverage large language models [15], [20] alongside formal methods to enhance the precision and efficiency of the analysis process. Large language models, such as GPT (Generative Pre-trained Transformer), have demonstrated remarkable capabilities in understanding and generating natural language text. When integrated into requirements analysis workflows, these models can assist in tasks such as requirements elicitation [14], documentation [20], ambiguity detection [13], [21], and incompleteness detection [15], [22]. By utilizing large language models, analysts can harness their ability to comprehend and contextualize complex requirements expressed in natural language, facilitating clearer communication and more accurate interpretation of stakeholder needs [14]. Moreover, large language models can aid in the generation of formal specifications from natural language requirements, streamlining the transition from informal to formal representations [23]. By combining the strengths of large language models with the rigor and precision of formal methods, recent works in requirements analysis aim to improve the accuracy, completeness, and efficiency of the analysis process, ultimately leading to more robust and reliable software systems [24].

B. TYPE OF DEFECTS IN REQUIREMENTS

Various research has proposed different taxonomies of defects that occur in the natural language of an SRS, often suggesting six defect types: ambiguous, incorrect, inconsistent, omission, superfluous, and conforming to standards [2]. However, three of these categories—ambiguity, inconsistency, and incompleteness—are commonly studied in the domain of automated defect detection due to their occurrence and predictability [14], [22]. Thus, our analysis focuses on these three primary tasks: identifying and resolving ambiguities, maintaining consistency across requirements, and ensuring the completeness of each requirement. The following sections will provide detailed discussions of these defects.

1) AMBIGUITIES

In requirements engineering, ambiguity refers to the potential for a requirement, expressed in NL, to be interpreted in multiple ways [25]. This can lead to different stakeholders understanding the same requirement differently. Generally, there are four types of ambiguity that might arise in the language of an SRS document [1], [26]. Lexical ambiguity arises when a word within the text has several meanings. Syntactic ambiguity occurs when a sentence or phrase can be grammatically structured in more than one way. Semantic ambiguity pertains to phrases that can be interpreted in multiple ways due to their verbal expressions within a given context. Finally, pragmatic ambiguity involves sentences that can have various interpretations or implications depending on the context in which they are read or understood.

2) INCONSISTENCIES

In the context of an SRS document, consistency issues occur when two or more requirements contradict each other. These contradictions can emerge for several reasons. For instance, during the requirements elicitation phase, different individuals might interpret requirements in varying ways, or the terminology used by those defining the SRS may differ from that used by developers [5], [27]. Additionally, inconsistencies may result when stakeholders have differing priorities for conflicting scenarios that haven't been adequately documented or discussed. Various types of inconsistencies can appear at different project stages [1]. For larger and more complex applications, it is crucial to resolve any inconsistencies within the SRS document before moving on to the design and development phases. Addressing these issues early in the project lifecycle is important because inconsistencies can lead to costly corrections later in the software system's development.

3) COMPLETENESS

According to IEEE guidelines, a requirements specification document should be comprehensive, meaning it should not contain any undefined information or details marked as "to be determined" [28]. In the field of RE, completeness is typically categorized into two types [27]. The first, internal completeness, pertains to the system's functions and qualities as deduced from the requirements document. The second type, external completeness, ensures that the Software Requirements Specification (SRS) encompasses all information suggested by external knowledge sources, such as stakeholders or related documentation (e.g., system descriptions). However, determining external completeness can be challenging because these external sources, including stakeholders, may not have a complete understanding of all the necessary requirements [15]. Generally, an SRS is considered complete when it includes all the essential information required for defining the problem and its solution. A complete and consistent requirement defines a correct requirement in the SRS document.

C. LLMS

Advancements in deep learning algorithms, coupled with the increase in available computational resources and large quantities of training data, have collectively fueled the rise of large language models (LLMs). LLMs represent a category of language models that leverage neural networks with billions of parameters, trained on extensive amounts of unlabeled text data using self-supervised learning techniques [29]. The origins of LLMs can be traced back to the early stages of language model and neural network development. Initial

attempts relied on statistical approaches and n-gram models, albeit with limitations in capturing long-term dependencies and contextual nuances in language. However, the rise of neural networks and the availability of larger datasets have contributed to the emergence of new, more intricate methods. The introduction of the Recurrent Neural Network (RNN) [30], which enabled the modeling of sequential data, including language, marked a pivotal milestone. However, RNNs faced challenges due to vanishing gradients and long-term dependencies. A significant breakthrough in LLM systems occurred with the introduction of the transformer architecture [31], which facilitated efficient handling of long-range dependencies through its self-attention mechanism.

The basic architecture pipeline of LLMs involves receiving text data from multiple sources and forwarding it to subsequent stages for preprocessing. The training process encompasses random parameter initialization, numerical data input, loss function calculation, parameter optimization, and iterative training. Once pre-trained on substantial corpora sourced from the web, these models have the capacity to grasp intricate language patterns, subtleties, and semantic connections. Despite their complexity, LLMs have demonstrated proficiency across various language-related tasks [32], including text synthesis, translation, summarization, question-answering, and sentiment analysis, thanks to the utilization of deep learning methodologies and large datasets. LLM architectures served as the foundation for models such as Google's Bidirectional Encoder Representations from Transformers (BERT) [33] and OpenAI's Generative Pre-trained Transformer (GPT) series [34], which excelled in various language tasks. Fine-tuning these models on specific downstream tasks has produced promising results, achieving state-of-the-art performance across several benchmarks. Previous research has highlighted the potential of LLMs in numerous Natural Language Processing (NLP) tasks, including specialized applications in domains such as medical and health sciences [35], education [36], and finance [37].

LLMs can learn through various paradigms, including zero-shot, few-shot, fine-tuning, and reinforcement learning, each presenting unique challenges, benefits, and costs [7], [38]. In zero-shot learning, LLMs tackle entirely new tasks without prior specific training, relying solely on their pre-existing knowledge. This approach enables the model to perform a wide array of tasks out-of-the-box, such as translation or summarization, without requiring additional data [39]. However, the challenge lies in achieving high accuracy and relevance, as the model may not fully grasp the task's nuances, leading to suboptimal performance [40]. Despite this, the cost is relatively low since no extra training data is needed [41]. Enhancements to LLM adaptability can be made through few-shot learning, where a small number of examples are provided for the model to learn from [42]. Fine-tuning LLMs involves adapting a pre-trained model to specific tasks using a smaller, domain-specific dataset. This process enhances the model's performance on particular applications without retraining from scratch. Fine-tuning updates the model's weights through gradient descent, optimizing for task-specific objectives. While it improves accuracy and relevance, the process comes with a high cost. One major issue is the substantial computational resources required, including significant processing power and memory, which can be costly and time-consuming. Additionally, fine-tuning demands large, high-quality datasets specific to the target domain, which are often difficult to obtain [43]. Fine-tuning can be further enhanced by incorporating Reinforcement Learning [44]. Reinforcement learning further refines these models by employing a reward-based system to iteratively improve their decision-making processes, but it requires higher costs to design an appropriate reward structure [45]. To tackle the cost challenge, several recent works have explored Low-Rank Adaptation (LoRA) [43]. Unlike traditional fine-tuning, which updates all the parameters of a pre-trained model, LoRA offers a more efficient approach by adjusting only a subset of parameters. LoRA achieves this by decomposing the weight matrices of the model into low-rank representations, enabling fine-tuning with significantly fewer parameters. This reduction in computational and storage costs makes LoRA particularly advantageous for LLMs, where traditional fine-tuning can be prohibitively expensive and resource-intensive. By focusing on a low-rank subspace, LoRA preserves the model's generalization capabilities while efficiently adapting it to new tasks or domains. This method ensures that LLMs can be fine-tuned and deployed in resource-constrained environments without sacrificing performance. Consequently, zero-shot learning often serves as a valuable benchmark for studies involving LLMs [46], [47].

The current top language models include OpenAI's GPT-4 [48], OpenAI's GPT-4o [49], Google's Bard [50], and Meta's LLaMA 2 [51]. Each model brings unique strengths to the field of natural language processing [52]. GPT-4 continues to serve as a benchmark with its extensive parameters and enhanced contextual comprehension. In contrast, GPT-4o pushes boundaries further by integrating multimodal capabilities, accepting text, audio, image, and video inputs, and generating diverse outputs with improved speed and cost efficiency. Meta's LLaMA 2 focuses on accessibility and efficiency, democratizing advanced AI with lower computational needs. Google's Bard, building on PaLM's [53] advancements, aims to deliver highly conversational and contextually aware interactions, particularly enhancing Google's search and assistant functionalities. Given its early success in this domain and the ease of access to its APIs, we have chosen to use OpenAI's GPT-4 model for our experiments [13], [14], [20].

III. METHODOLOGY

A. OBJECTIVE

The objective of this research is to evaluate the effectiveness of GPT-4 in performing requirements analysis

tasks. Analyzing requirements involves understanding and evaluating the gathered requirements to ensure the creation of high-quality SRS documents. The primary challenge in this step is to identify ambiguous, inconsistent, and incomplete requirements written in natural language. This study aims to establish a baseline for assessing the performance of GPT-4 in detecting defects across these three categories. To develop these baselines, we select an industrial case study (further detailed in Section III-B), which includes multiple components and a hierarchical structure within the SRS. Assessing each requirement in isolation can lead to a significant number of false positives, as requirements are often interconnected, particularly within the same component or section. Moreover, practical SRS documents include diagrams and figures that must be evaluated alongside the text to accurately assess each requirement's quality. Thus, our case study is designed to provide a comprehensive evaluation of the effectiveness of LLM models when applied to a well-structured SRS document that includes various components and figures.

B. CASE STUDY

The case study we chose to evaluate our methodology is based on the Mechanical Lung Ventilator (MLV) presented by the ABZ 2024 conference organizers [16]. This system is designed to aid patients who struggle with breathing due to various health issues, operating under two modes: Pressure-Controlled Ventilation (PCV) and Pressure-Support Ventilation (PSV). The goal of using the MLV case study from the ABZ conference is to scrutinize the software specifications provided and to model the system for formal verification of its operational integrity. We can analyze the system's behavior through formal modeling to ensure its correctness. During this process, the Software Requirements Specification (SRS) document for the MLV system revealed several ambiguities or unclear requirements, highlighting the importance of modeling for identifying such issues.

The document on the MLV utilized in our research is an extensive industrial document that presents significant challenges for manual review, unlike the examples in prior studies. This document spans 52 pages, and it is organized into five main sections featuring both textual and visual descriptions. These sections include (1) an introduction, which outlines the history and objective of the MLV; (2) system requirements, detailing the overall system and its requirements; (3) GUI (Graphical User Interface) requirements; (4) controller requirements; and (5) alarms, which are triggered by either the controller or GUI to signal errors and ensure patient safety. The GUI and the controller represent key subsystems of the MLV. Requirements in each section are articulated in natural language and are supplemented with diagrams. The comprehensive requirements for the MLV system can be found at https://github.com/foselab/abz2024_casestudy_MLV. Table 1 showcases a selection of requirements from different sections of the document, including System Requirements (identified by IDs starting with FUN, PER, and SAV), GUI specifications, Controller specifications (identified by IDs starting with CONT), and Alarm specifications (identified by IDs starting with AL).

TABLE 1. Examples of requirements from the MLV SRS document.

ID	Requirement
FUN.8.5	The system shall not permit the healthcare professional operator to erase the contents of the alarm system log. (ISO 80601-2-12)
FUN.18	The system shall have a leak compensation feature for leaks in the patient breathing circuit which shall be disabled by default.
FUN.48.1	The user shall set the desired FiO2 value from which the +-3% alarm limits are derived. The input FiO2 value will have to be manually adjusted by the user until the desired FiO2 value is displayed
PER.14	Target inspiratory pressure (P_insp_AP) | Default 0 cm H2O | Range 2-50 cm H2O | Step Size 1 cm H2O
SAV.24.1	The alarm condition delay for high PEEP alarm condition shall not exceed the duration of three inflations.
GUI.1.2	Start Mode: allows the user to resume ventilation or to start the ventilation for a new patient.
GUI.4	The transition from Start to Self Test shall occur if the user wants to test the machine and set the proper parameters for a new patient
GUI.45.2	When resuming ventilation, the GUI shall be able to load setting parameters from the last known configuration saved by the system and stored in the system. This configuration is protected by a md5 file to guarantee that the settings are not corrupted. Before loading, the GUI checks the integrity of the file.
CONT.5	The transition from VentilationOff to PSV shall occur if the change mode command is received from the GUI.
CONT.6	The transition from VentilationOff to PCV mode shall occur if the change mode command is received from the GUI.
AL.1	The user shall be able to set alarm thresholds when the ventilator is either ventilating or not.
AL.9	The system shall prevent the operator from saving changes to the alarm preset.
AL.5	Additional requirements for 1 m (operator's position) visual alarm signals and information signals High priority alarm signals should be accompanied by information describing possible causes of the alarm condition and appropriate actions to take in response. Operator action may be required to display this information.

C. DATA PREPARATION

We employed the OpenAI API to streamline the analysis process for our experiments. We utilized the GPT-4 Turbo model to assess its performance. Additionally, the GPT-4 Vision model was utilized to extract information from graphical figures. The temperature was set to 0 for all experiments. Setting the temperature to 0 in LLMs maintains consistency by making the model's output deterministic. The temperature parameter in LLMs controls the randomness of the predictions. When the temperature is set to 0, the model effectively always picks the token with the highest probability, leading to a consistent and repeatable output for a given input. This deterministic behavior ensures that the model generates the same response every time it encounters the same prompt, which guarantees reliable and uniform

responses [54]. In the following sections, we will elaborate on the preparation of text data and the extraction of information from figures for integration with API calls.

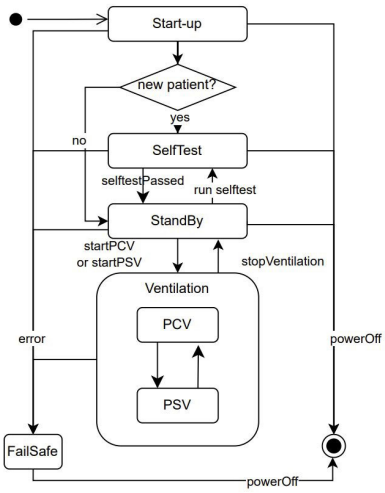
1) TEXT EXTRACTION

In preparing the textual content from the PDF document of the MLV requirements, we undertook three critical procedures: data cleaning, data formatting, and eliminating extraneous information. The removal process focused on excluding version histories and repeated section titles that might lead to confusion in the LLM. During the data cleaning phase, we addressed several issues. Notably, when transferring text from the PDF, numerous special characters, such as semi-colons, were incorrectly represented. These characters were identified and corrected to ensure accuracy. Additionally, page numbers were removed to avoid introducing numerical errors into requirements that contained quantitative details. We also standardized various inconsistently transferred abbreviations, such as "RR_AP" appearing as 'RRAP' or 'RR AP' to ensure uniformity and facilitate correct interpretation in conjunction with the abbreviations table. Regular expressions (Regex) were employed extensively for text cleaning, supplemented by thorough manual reviews to guarantee that the text accurately reflected the document's contents. Regarding data formatting, we worked to accurately represent tables by inserting hyphens to delineate columns, preserving the semantic relationship between field headings and their corresponding values. This effort was crucial to maintain the integrity of the table structures in a text format.

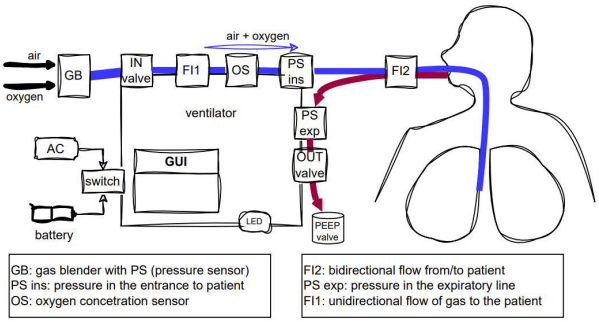
2) FIGURE DESCRIPTION

The SRS document for the Mechanical Lung Ventilator (MLV) system included six figures. Among these, four figures depicted standard UML diagrams illustrating various aspects of the system or its components. The two nonstandard diagrams were Figure 2.2 (presented in Fig. 1) from the Systems Requirements in the SRS document, showcasing the sensors' interfacing of the MLV system, and Figure 3.2 from the GUI section illustrating a draft interface of the system. To facilitate the extraction of detailed descriptions from these figures, we initially converted the images into base64 encoding. Subsequently, the GPT Vision (GPT-V) API was employed to interpret and describe the content of each figure. The instructions provided to the GPT-V API were structured as follows: "Given the UML diagram of the <caption> of a mechanical lung ventilator, provide a descriptive prompt that exactly describes the UML diagram. Make sure to briefly mention the figure label in the beginning. Provide the descriptions in well-defined points." The <caption> utilized in each request was directly derived from the SRS document to ensure accuracy and relevance in the model's responses. Figure 1 presents some of the sample figures from the SRS document.

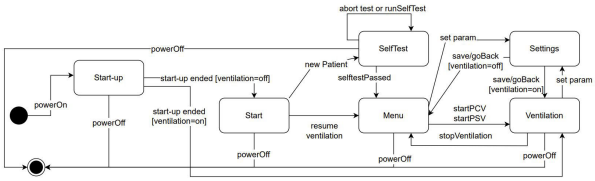
The descriptions generated for each figure contained several inaccuracies, necessitating manual corrections. Notably, the GPT-V model often misinterpreted the direction of arrows



(a) High level operation diagram



(b) High level view of ventilator sensors



(c) GUI state machine

FIGURE 1. Figure samples from the SRS document [16], where (a) is Figure 2.1, (b) is Figure 2.2, and (c) is Figure 3.1 in the actual document.

in diagrams. For instance, an error pertaining to the overall system was identified and corrected in Figure 1, where a one-way arrow between the sensors and actuators of the MLV was inaccurately depicted as bidirectional. Similar inaccuracies were observed in the flow of state diagrams; for example, the GUI state machine diagram contained errors in the transition of arrows from the Settings state to other states upon the set_param event. Additionally, GPT-V inaccurately represented the capability of a state to loop back to itself upon a specific event; in the GUI state machine, the SelfTest state should remain active if the 'abort test' event is triggered, yet this was not correctly described. Omissions in event transitions were also noted, such as the missed transition from PSV to PCV upon the apneaLag event. However, it is worth mentioning that GPT-V provided a relatively

accurate description of the unconventional representation in Figure 2.2, which depicted the connections between sensors with wiggly lines despite minor textual inaccuracies. Overall, it is observed that employing GPT-V for initial image description generation, followed by a detailed review and correction of these descriptions, significantly accelerates the process compared to composing image descriptions from the ground up. The full source code, outputs, and figure descriptions for our project are accessible via the GitHub link: https://github.com/Taslim-M/GPT4-Requirements-Analysis.

D. PIPELINE FOR REQUIREMENTS ANALYSIS

For our study, we systematically fed extracted text segments and figure descriptions into the GPT APIs, employing the GPT-4 Turbo models for comprehensive analysis. Our objective was to leverage the LLM to identify any ambiguities, inconsistencies, and incompleteness of the MLV requirements. Each segment within the five primary sections of the Software Requirements Specification (SRS) document—Introduction, System Requirements, Graphical User Interface (GUI), Controller, and Alarms—was processed individually. Figure 2 shows the overall pipeline followed in our methodology.

We crafted specific prompts for each section to pinpoint problematic requirements within every subsection and table of the primary sections. To discover ambiguities, we used a standardized prompt format: "Are there any ambiguities or unclear statements within the <sub-section>? (Evaluate any ambiguities or unclear statements within the <sub-section>.)" Here, "<sub-section>" denotes the specific sub-headings extracted from the SRS document. For assessing completeness, we applied a prompt structured as: "Is the <sub-section>, including the fields for ID, Requirement/Rationale, and Input Reference, complete? (Assess whether all required fields are adequately filled out and whether any crucial details are missing from the <sub-section>.)" Similarly, to detect inconsistencies, we utilized: "Are there any inconsistencies and/or contradictions within the <sub-section>? (Evaluate any inconsistencies or contradictions within the <sub-section>.)" After conducting these detailed, section-by-section evaluations, we further analyze the impact of providing global context to the GPT-4 model. This secondary analysis aims to determine two key aspects: (1) whether the token limitations inherent to LLM models, such as GPT-3.5, affect the quality of defect detection, and (2) whether additional context can mitigate the false positives that arise from insufficient information available to the LLM. The prompts were crafted through discussions among the authors, focusing on the language and style typically employed by industrial practitioners. Several prompts were proposed, and the final one was designed to mirror what we considered to be a typical style of direct prompting.

1) EVALUATION CRITERIA

The responses generated by the GPT-4 model were compiled into a CSV file for systematic evaluation based on predefined criteria. Two authors (T.M. and D.D.) independently assessed the defects identified by the LLM. Following individual assessments, they discussed their feedback to resolve any discrepancies and reach a consensus decision. This collaborative approach helps mitigate biases and enhances the reliability of the assessment, particularly in evaluating the importance of any identified defects.

Initially, we categorized the correctness of the responses as either True (accurate detection) or False (inaccurate detection). For responses classified as True, indicating a correctly identified error (ambiguity, inconsistency, or incompleteness), we further evaluated the severity or importance of these errors on a scale from 1 to 5. This severity scale is calibrated based on the potential impact of the identified requirement flaw on the system's design or its essential functionalities. A severity rating of 1 signifies negligible impact, indicating that the identified error, while accurate, holds minimal importance. Conversely, a rating of 5 indicates a critical flaw that could lead to significant conflicts or issues within the system design.

For errors inaccurately flagged by the GPT-4 model, we conducted an additional layer of analysis to determine the underlying cause. Specifically, we examined whether the misidentification stemmed from a lack of domain-specific knowledge or was a consequence of the LLMs' limitations in processing information from other sections of the document, possibly due to token constraints. This two-pronged approach to evaluating the LLMs' responses does not only help quantify the accuracy and relevance of detected errors but also provides insights into the operational limitations of the models and areas for improvement in processing complex technical documents.

The experiment was also conducted separately for the diagrams presented in the MLV SRS document, focusing exclusively on detecting ambiguities. The assessment criteria used for individual requirements were applied, with the addition of a 'Syntax Limitation' field. This field, marked as either true or false, indicates whether a false positive defect identified by the LLM is attributable to its inability to comprehend the nuances of diagrammatic descriptions in the figures.

2) METRICS REPORTED

The True Positive and False Positive defects were first qualitatively analyzed for each of the three categories of defects. For both the correct and incorrect defects, we categorized them into meaningful operational sections to highlight areas where the GPT-4 performs well and where they commonly produce false positives. Quantitatively, we measured the precision of the identified defects for each category. Precision was calculated using Eq 1, providing a clear metric to evaluate the LLM's performance in detecting ambiguities, inconsistencies, and incompleteness.

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

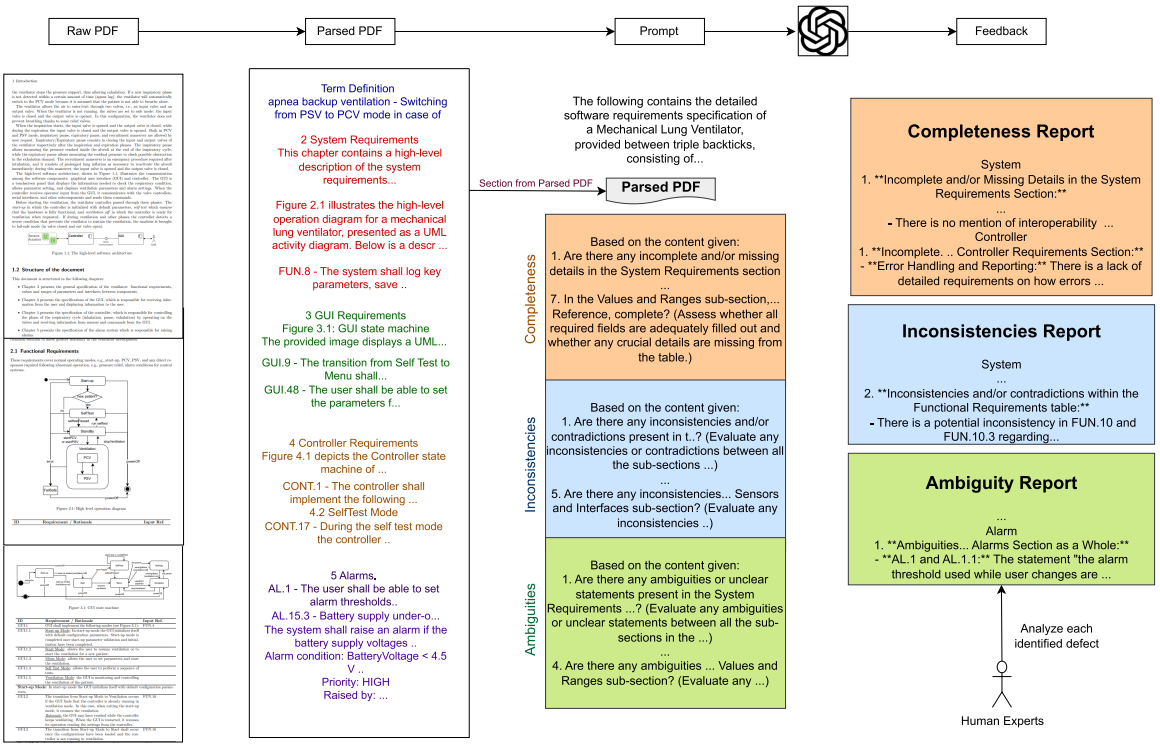


FIGURE 2. Pipeline for generating LLM feedback on requirement analysis using the GPT-4 model. Given a PDF, we parse and extract the document's contents into different sections, including figure descriptions. We then prompt the GPT model to provide structured comments for each defect category. The prompt is fed with context that includes a complete section (e.g., System, Controller, etc.) from the parsed PDF to ensure the input is within the token constraints of the GPT model. The GPT models generate feedback in a single pass, structured across the three defect categories (ambiguities, inconsistencies, incompleteness) and the various SRS sections. Human experts analyze the feedback according to the predefined evaluation criteria.

where *TP* represents the number of True Positives and *FP* represents the number of False Positives. Here, we do not have the exact number of defects per category, and formal modeling can overlook certain NL-based defects. Therefore, we did not establish a comprehensive ground truth, and our metric does not include a recall value.

IV. RESULTS

A. AMBIGUITIES

We provide a comprehensive analysis of the detected defects in the software requirements, structured into qualitative and quantitative segments. Initially, we provide a qualitative examination of the defects, detailing the nature and implications of each identified issue. Subsequently, we shift focus to quantitative analysis, specifically evaluating the true positive and false positive outcomes of our detection methods. While our discussion does not prioritize the types of ambiguities based on their severity, we thoroughly assess the overall accuracy of the identified ambiguities.

1) QUALITATIVE ANALYSIS

Tables 2 and 3 present a selection of the GPT-4 model's detections of ambiguous requirements. The correctly identified examples (Table 2) highlight true positive detections by the LLM, spanning categories such as system operation and safety, system workflows, and identifying issues in the range of values for system variables. In contrast, the false positive examples (Table 3) typically occur in technical graphical requirements. These false positives are often due to the LLM's limited domain knowledge and its inability to process information beyond a specific context window.

In the analysis of ambiguities correctly identified by the GPT-4 model, as documented in Table 2, several salient observations emerge. The LLM adeptly identifies ambiguities involving missing numerical calculations or system equations, which are crucial for the proper implementation of system operations. For instance, in example SYS.GPT4.10, the GPT successfully detects the absence of a direct mathematical formula required for calculating the inflation duration. Resolving this ambiguity necessitates an additional interpretation of graphical data, which is located elsewhere in the SRS. Moreover, the GPT-4 model demonstrates a keen ability to comprehend the expected range of values for variables and to identify conflicts when the default value specified in the SRS falls outside this range, as illustrated in SYS.GPT4.7. This capability is not only indicative of the LLM's precision in handling numerical data but also its potential to ensure the consistency and reliability of system specifications.

The GPT-4 model also demonstrates considerable efficacy in identifying ambiguities that are crucial for the operational safety of the mechanical lung ventilator (MLV). Notably,

TABLE 2. Illustrative samples of correctly identified ambiguities using GPT-4 model.

ID	Defect Info
SYS.GPT4.10	SAV.24.1 mentions that the "alarm condition delay for high PEEP alarm condition shall not exceed the duration of three inflations," but it's unclear how "the duration of three inflations" is measured or determined, potentially leading to variability in implementation.
SYS.GPT4.3	FUN.18 states the leak compensation feature "shall be disabled by default," but does not specify how or when it can be enabled, or under what conditions it should be used.
SYS.GPT4.7	PER.14 has a default value of 0 cm H2O for P_insp AP, which seems counterintuitive since this setting is critical for apnea backup settings. The rationale for leaving it unset is unclear and could lead to potential misuse or confusion.
GUI.GPT4.6	GUI.45.2 mentions loading settings from a "last known configuration" protected by an md5 file but does not specify how frequently these configurations are saved or under what conditions, potentially leading to uncertainty about the freshness of these settings.
GUI.GPT4.11	GUI.61.2's requirement for confirmation/setting of parameters upon a mode switch could be clearer regarding the user interface flow and how the confirmation is sought.
ALR.GPT4.1	AL.1 and AL.1.1: The statement "the alarm threshold used while user changes are the last saved" is somewhat ambiguous. It's unclear whether it means the last saved thresholds are used during the change process or immediately after changes are saved.
ALR.GPT4.2	AL.5: The requirement that "Operator action may be required to display this information" is vague. It does not specify what action is required or under what circumstances this action is needed.

TABLE 3. Illustrative samples of incorrectly identified ambiguities defects using GPT-4 model.

ID	Defect Info
SYS.GPT4.4	FUN.20 and FUN.22 mention specific modes and procedures but lack detail on how these should be executed or under what conditions they are allowed or recommended.
SYS.GPT4.5	FUN.48.1 states the user must manually adjust the FiO2 value until the desired value is displayed, but it does not specify how the user is informed of the current value or the mechanism for adjustment, which could lead to usability issues.
SYS.GPT4.9	SAV.14 and SAV.15 mention conditions for leakage and obstruction alarms but do not specify how these conditions are detected or the threshold for what constitutes "significant" leakage or obstruction.
GUI.GPT4.5	GUI.13 lacks a clear requirement statement. It introduces a rationale without explicitly stating the requirement, leading to ambiguity about what the GUI should do when the controller is already ventilating upon GUI start.
GUI.GPT4.13	GUI.103's requirement for shifting and re-scaling frozen waveforms could benefit from more detail on the expected functionality or user interaction for these actions.
CONT.GPT4.5	The use of terms like "if required by the GUI" (e.g., CONT.43) without specifying the conditions under which these requirements are triggered could lead to different interpretations of the requirements.

it successfully detects vague language within the requirements that could significantly affect the system's safety, as evidenced in the example ALR.GPT4.1. However, it is important to acknowledge that the LLM failed to recognize another ambiguity within the same requirement, highlighting an area for potential improvement in its detection capabilities.

Additionally, the GPT-4 model is adept at interpreting the system's workflow and identifying critical ambiguities. For instance, it pinpointed an ambiguity in the file-saving operation of system configurations, a process of paramount importance in a medical device like the MLV (referenced in GUI.GPT4.6). The ability of the LLM to track and potentially enhance the graphical user interface flow further underscores its utility, as seen in its suggestions for improvements in GUI.GPT4.11. Moreover, the GPT-4 model's capacity to scrutinize individual requirements and detect ambiguities related to vague language—mirroring the function of traditional rule-based NLP detectors—is illustrated in examples such as SYS.GPT4.3 and ALR.GPT4.2.

The significance of these true positive findings varies based on their potential impact on system functionality. Ambiguities such as those found in SYS.GPT4.3, SYS.GPT4.10, and GUI.GPT4.11, which can either be resolved by a thorough document review or considered less critical to system behavior, receives a severity rating of 2/5. More pivotal issues like those in SYS.GPT4.7, GUI.GPT4.6, and ALR.GPT4.2, which illustrates substantial gaps in system operation instructions, are rated at 3/5, reflecting their greater importance in ensuring system integrity. We assign a severity rating of 4/5 to ALR.GPT4.1 due to the ambiguity that persisted even after multiple reviews by human experts.

In our analysis of false positive results, a substantial number can be traced back to the LLM's challenges in comprehending both local and global contexts. For example, SYS.GPT4.4 demonstrates the LLM's oversight of related sub-requirements found in FUN 21 and 23, which indicates a gap in the model's ability to link information across different sections. Additionally, GUI.GPT4.5 incorrectly flags ambiguities about the expected behavior of the GUI during startup ventilation, despite clear guidelines in GUI 13.1.

The issue of global context limitation is further exemplified in SYS.GPT4.5, where the LLM fails to acknowledge that the FiO2 value details are comprehensively addressed in both PER 2 and GUI 89, leading to an unwarranted ambiguity flag. Similarly, SYS.GPT4.9 shows the LLM mistakenly identifying the term 'significant' as ambiguous in the context of an alarm for leakage, missing the clarifications provided by examples from ISO standards. These instances highlight the critical need for LLMs to possess robust data processing capabilities to discern contextual nuances and reduce false positives.

The lack of domain knowledge and contextual understanding in the model is exemplified by GUI.GPT4.13, which highlights a perceived ambiguity related to waveform shifting functionalities. This ambiguity is particularly significant as requirements GUI 73-75 emphasize the criticality of waveform functionalities in monitoring patient vitals. The LLM's inability to grasp this nuance underscores the

importance of enhancing its domain-specific and contextual awareness.

Despite some other trivial false positives, the LLM's findings often prompt valuable suggestions for enhancing the clarity of the SRS. Many false positives, initially examined by human experts and deemed incorrect, have nevertheless highlighted areas where the presentation of information could be optimized. For instance, CONT.GPT4.5 ends ambiguously with 'if required by the GUI', prompting a deeper review of both the overall system and GUI sections for comprehensive understanding. Therefore, while many false positives are ultimately dismissed, they serve as catalysts for refining the articulation of requirements, thereby potentially elevating the quality of the SRS.

2) QUANTITATIVE METRICS

Table 4 highlights the performance of the GPT-4 model in detecting ambiguities across the different components of the SRS.

TABLE 4. Precision across the different categories for detecting ambiguities.

Component	True Positive (count)	False Positive (count)	Precision
System	6	4	0.60
GUI	5	7	0.41
Controller	2	10	0.17
Alarm	3	5	0.38

In our quantitative evaluation of the System Requirements section, the GPT-4 model identified six true positive ambiguities and four false positives, yielding a precision of 0.6. Half of the incorrect identifications were linked to deficiencies in domain knowledge, while all four false positives stemmed from context limitations—split evenly between global and local contexts.

In the GUI section, our analysis revealed five true positives and seven false positives, resulting in a precision rate of 0.42. Three of these false positives were due to context limitations, and one resulted from a lack of domain knowledge. Additionally, two false positives were caused by ambiguous row segmentation in the GUI requirements table—a detail typically overlooked by human experts.

In the Controller section, we noted the lowest precision rate at 0.17, with only two correct detections against ten false positives. Seven of these false positives were due to context limitations, and five related to gaps in domain knowledge.

The Alarm section analysis resulted in three true positives and five false positives, achieving a precision of 0.375. Among the false positives, three were related to context limitations, and two were due to insufficient domain knowledge.

Figure 3 depicts the distribution of the importance levels for all identified ambiguities. The majority of these ambiguities (11 out of 16) fall into the low-significance categories, with severity ratings of 1 or 2. Four ambiguities are classified as moderately important, with a severity rating of 3/5. Finally, one ambiguity is marked as significant for system specifications, with a severity rating of 4/5. This distribution indicates that most of the identified ambiguities have a low to moderate impact, while a smaller number are more critical to system performance and compliance.

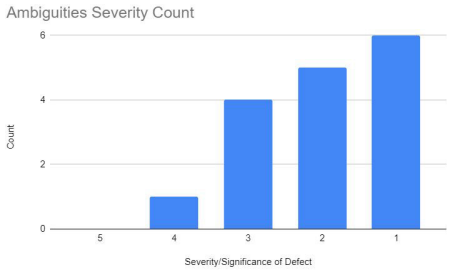


FIGURE 3. Severity count of ambiguities.

3) DOES GLOBAL CONTEXT IMPROVE LLMS FEEDBACK?

In our efforts to integrate a global context into the analysis, we conducted section-wise checks for ambiguities. When the prompts were generalized to cover entire sections (such as Controller, GUI, etc.) without delving into specific sub-headings, we observed a decline in the LLM's detail orientation. In this approach, only four ambiguities were flagged, with two being accurately identified, resulting in a precision of 0.5. Although this method allows for quicker analysis, it becomes evident that it is crucial to employ section-specific prompts to achieve a thorough quality check and effectively utilize the GPT-4 model.

To maintain a consistent comparison, we used the section-wise prompts previously employed with the GPT-4 model but included the full context of the SRS to reduce false positives caused by missing information outside a specific section. Despite this, we observed a significant overlap in the analysis, indicating that the GPT-4 model does not yet match human capabilities in meticulously cross-referencing requirements. Notably, 6 out of 7 false positives initially attributed to a lack of global context persisted even with the full context provided to the LLM. Using the Set Overlap ratio, we found an overall 0.61 overlap between the section-wise context and the global context, with many significant defects not being recalled in the global context. This highlights a limitation in the LLM's ability to fully integrate and apply the broader context, underscoring the need for more sophisticated contextual handling in the model.

B. INCONSISTENCIES

1) QUALITATIVE ANALYSIS

Tables 5 and 6 showcase selected examples of inconsistent requirements identified by the GPT-4 model. The true positive detections (Table 5) demonstrate that LLMs are proficient in identifying operational and language-based inconsistencies. However, the false positive examples (Table 6) indicate

TABLE 5. Illustrative samples of correctly identified inconsistencies defects using GPT-4 model.

ID	Defect Info
SYS.GPT4.3	FUN.8.5 states that the healthcare professional operator shall not erase the contents of the alarm system log, which contradicts FUN.8.4 that implies the log can be erased by authorized personnel. The distinction between "healthcare professional operator" and "authorized personnel" needs clarification to avoid confusion.
SYS.GPT4.5	PER.14 (Target inspiratory pressure (P_insp AP)) has a default value of 0 cm H2O, which seems inconsistent with the range of 2-50 cm H2O. This could be a typo or oversight, as a default value outside the allowed range is not logical.
ALR.GPT4.1	There is a potential inconsistency between AL.1 and AL.9. AL.1 allows the user to set alarm thresholds, implying that changes can be saved, while AL.9 states that the system shall prevent the operator from saving changes to the alarm preset. This could be interpreted as contradictory unless "alarm preset" and "alarm thresholds" are defined as distinct entities within the system, with only the preset being locked from changes.

TABLE 6. Illustrative samples of incorrectly identified inconsistencies defects using GPT-4 model.

ID	Defect Info
CONT.GPT4.1	- There is a contradiction between CONT.5 and CONT.6 regarding the rationale for transitioning from VentilationOff to PSV or PCV mode. CONT.5 states the ventilation starts in PCV mode when the user selects the start command from the GUI, while CONT.6 states the ventilation starts in PSV mode when the user selects the start command from the GUI. This is contradictory as it suggests both PSV and PCV modes are the starting modes for ventilation.
GUI.GPT4.3	The "Self Test Mode" descriptions (GUI.4 and GUI.9) suggest that the self-test is a prerequisite for entering the "Menu" state, but the UML diagram shows a direct transition from "Start" to "Menu" without mentioning the self-test.

that LLMs struggle with identifying inconsistencies related to diagrams and comprehending complex system flows.

The GPT-4 model has demonstrated a strong ability to detect inconsistencies in software operations. A prime example is in SYS.GPT4.3, where the LLM identifies a lack of clarity in access control requirements. The inconsistency arises from the use of terms like 'authorized personnel' and 'healthcare professional operator,' which are not clearly defined in the SRS. This ambiguity could lead to conflicts during the software's implementation stage. Detecting such defects can significantly improve the quality of the SRS by clarifying the roles and responsibilities of different system users.

Moreover, GPT-4 is useful for identifying inconsistencies in the language of the requirements. For instance, the system requirements state that users can set alarm thresholds, but they also prevent operators from saving changes to the alarm preset. This inconsistency stems from the SRS not explicitly differentiating between 'alarm threshold' and 'alarm preset.' Additionally, the LLM, owing to its language analysis capabilities, can spot inconsistencies in default values that fall outside the acceptable range, as observed in ALR.GPT4.1.

In examining the incorrectly identified defects, it is apparent that GPT-4 demonstrates a reasonable ability to correlate UML diagrams with the SRS and cross-check them against requirements. Although one of the identified defects (GUI.GPT4.3) did not represent a true inconsistency, this outcome is still notable, as it suggests the LLM's potential to detect interactions between diagrams and requirements in large industrial systems. However, our findings indicate that the LLM does not always verify the flow of UML diagrams against the requirements with complete accuracy, leading to false positives in inconsistency detection. In some cases, the LLM failed to discern between different system flows when analyzing local contexts. This occurs when multiple requirements overlap but suggest different pathways within the system, causing the LLM to flag a defect when, in fact, the variations represent legitimate alternative flows.

These incorrect false positives underscore the LLM's limitation in understanding the flexible nature of complex systems, where multiple flows can originate from a single point. This indicates that while LLMs can be valuable in cross-referencing diagrams and requirements, further refinement is needed to reduce the occurrence of erroneous inconsistencies.

2) QUANTITATIVE METRICS

In the quantitative evaluation of inconsistencies detected by the GPT-4 model, there are varying levels of precision across different subsystems, as depicted in Table 7. In the System Requirements section, the LLM performs well, with 4 true positives and 3 false positives, yielding a precision of 0.57. Of these, two were significant, with importance ratings of 5 and 4, respectively.

TABLE 7. Precision across the different categories for detecting inconsistencies.

Component	True Positive (count)	False Positive (count)	Precision
System	4	3	0.57
GUI	2	2	0.50
Controller	1	2	0.33
Alarm	1	2	0.33

In the GUI section, the precision is 0.5, with 2 correct and 2 incorrect inconsistencies. These inconsistencies are of lower importance since they do not relate to system functionality or behavior but rather to the arrangement and order within the SRS.

In the Controller section, the precision is also 0.5, with 2 false positives. These false positives are linked to GPT-4's limited ability to connect related requirements within a localized context.

In the Alarms section, the precision drops to 0.33, with only 1 correct out of 3 defects. One of the false positives is due to the GPT-4 model misinterpreting alarm requirements for two different sensors, highlighting a common issue where the

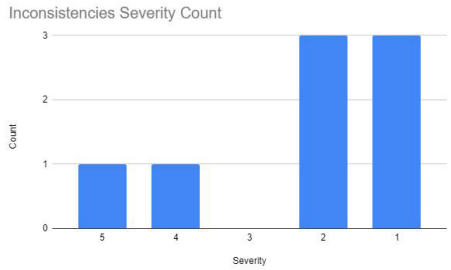


FIGURE 4. Severity count of inconsistencies.

LLM tends to group requirements based on textual content without differentiating between components.

These observations suggest that while GPT-4 can effectively identify inconsistencies in some cases, it struggles with contextual understanding, leading to false positives when the requirements have similar text patterns but pertain to different components. Further refinement in contextual analysis and differentiation of similar-sounding requirements could improve the LLM's precision in detecting inconsistencies.

Figure 4 illustrates the distribution of severity ratings for all identified inconsistent defects. The majority of ambiguities are of low importance, with 6 out of 8 falling into the lower severity categories of 1 or 2. However, there are two notable exceptions, classified as high-importance ambiguities, with severity ratings of 4 and 5, respectively. These higher-rated ambiguities are likely to have a more substantial impact on system functionality or compliance.

C. COMPLETENESS

1) QUALITATIVE ANALYSIS

Tables 8 and 9 present selected examples of incomplete requirements identified by the GPT-4 model. The true positive completeness defects (Table 8) illustrate that LLMs can identify incompleteness related to system safety, privacy, and industry standards. Conversely, the false positive defects (Table 9) highlight that the GPT-4 model lacks the intuitive ability of humans to fully comprehend and complete requirements, often failing to explore the entire provided context.

TABLE 8. Illustrative samples of correctly identified completeness defects using GPT-4 model.

ID	Defect Info
ALR.GPT4.2	Auditory Alarms: The section focuses on visual alarms and their specifications but does not mention auditory alarms. According to IEC 60601-1-8, medical devices should also have auditory alarms for certain conditions, especially for high-priority alarms.
GUI.GPT4.2	Security requirements for the GUI, especially regarding patient data protection and access control, are not specified.
SYS.GPT4.3	The requirements for user authentication, authorization, and audit trails are not mentioned, which are critical for patient data protection and compliance with regulations like HIPAA in the US.

TABLE 9. Illustrative samples of incorrectly identified completeness defects using GPT-4 model.

ID	Defect Info
SYS.GPT4.4	The table lacks default values for some parameters (e.g., PER.1 and PER.2), which are essential for initial device setup and ensuring safe operation within predefined limits.
ALR.GPT4.3	Alarm Log or History: The specification does not mention if there is an alarm log or history feature that records when each alarm was triggered and acknowledged.
CONT.GPT4.3	The requirements do not specify the behavior of the system in case of power failure, which is a critical scenario for a life-supporting device.

GPT-4 has demonstrated significant competence in suggesting improvements for the completion of SRS, particularly in identifying gaps that could affect the safety of the MLV system. For instance, the LLM identified missing information regarding different types of system users, their roles, and authentication protocols (ALR.GPT4.2). Additionally, it utilized basic domain knowledge of GUIs to point out that the SRS lacks details about patient data visibility to ensure privacy.

GPT-4 also exhibited a notable ability to cross-reference requirements against industry standards. Even without explicit prompting, the model sometimes checked requirements against relevant standards. In ALR.GPT4.2, for example, the LLM flagged the absence of a requirement for auditory alarms in compliance with IEC 60606-1-8, underscoring the importance of auditory signals in emergency scenarios where visual alarms might not be adequate. Furthermore, LLMs can use their understanding of regulatory requirements to suggest enhancements to the SRS. In SYS.GPT4.3, the GPT-4 model pointed out the absence of user authentication details, referencing HIPAA compliance in the United States as a critical reason for strengthening security.

Regarding the significance of identified issues, both ALR.GPT4.2 and SYS.GPT4.2 are rated 4/5, indicating they suggest critical improvements for system security and regulatory compliance. GUI.GPT4.2 is rated 2/5, indicating a moderate improvement related to privacy in GUI displays. This lower rating reflects the understanding that MLV-use settings like hospitals and clinics already have robust security measures to control who can access patient data.

In contrast, when examining false positives, the GPT-4 model showed limitations in logically interpreting some requirements compared to human readers. For example, the LLM failed to recognize that specific values, like default values for PEEP, are derived from external valve sensors even though these defaults were not explicitly mentioned in the SRS (SYS.GPT4.4). Additionally, a lack of global context led the LLM to incorrectly flag some defects as incomplete, as seen in ALR.GPT4.3 and CONT.GPT4.3. Moreover, human readers can intuitively fill in certain gaps that the LLM deems incomplete, such as when the LLM

TABLE 10. Precision across the different categories for detecting incompleteness.

Component	True Positive (count)	False Positive (count)	Precision
System	3	2	0.60
GUI	2	1	0.67
Controller	2	2	0.50
Alarm	2	1	0.67

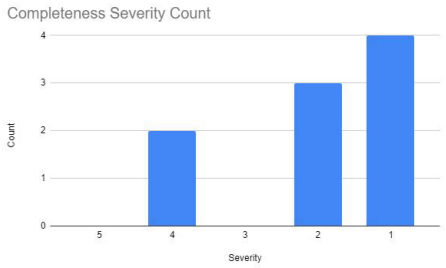


FIGURE 5. Severity count of completeness.

flagged missing regulatory document versions as a defect, even though it's generally understood that the latest versions should be used based on the implementation date. This indicates that while LLMs can be helpful, they require further refinement to match human-like intuitive understanding and contextual interpretation.

2) QUANTITATIVE METRICS

Table 10 illustrates the performance of the GPT-4 model in identifying completeness defects across different categories. Overall, except for the Controller section, all other sections show a precision rate of over 50% in identifying completeness defects. In the System Requirements section, the LLM identified 3 true positive (TP) completeness defects and 2 false positive (FP) defects. Both false positives are attributed to a lack of global context, and one is linked to insufficient domain knowledge.

In the GUI section, II defects were identified as true positives, with 1 false positive. The false positive resulted from incomplete figure descriptions, indicating a limitation in consolidating local context before flagging a completeness defect.

In the Controller section, GPT-4 detected 2 true positives and 2 false positives, with the latter resulting from a lack of global context when evaluating the requirements. Finally, in the Alarm section, II true positives and 1 false positive were recorded. The false positive here is due to the LLM's inability to incorporate global context, leading to an incorrect classification of a requirement as incomplete.

These results suggest that the GPT -4 model tends to perform well in identifying completeness defects when contextual information is sufficiently provided, but it may struggle with false positives when there are gaps in a global context or domain-specific knowledge.

TABLE 11. Samples of correctly identified defects in Figures using GPT-4 model.

ID	Defect Info
FIG3.GPT4.2	While it's mentioned that all states besides the initial, final, and Settings states have a transition to the final state when the "powerOff" event occurs, it's ambiguous why the "Settings" state is excluded from this rule. This could be an oversight or imply a specific behavior that is not clearly explained.
FIG2.GPT4.2	It's mentioned that the system transitions to the "StandBy" stage both after a successful self-test (when a new patient is detected) and directly if the answer is 'no' at the '<new patient?>' decision point. However, it's not clear if the conditions or the system's state in "StandBy" are different based on these two paths.
FIG4.GPT4.2	The description uses both 'error' and specific commands like 'startPCV', 'startPSV', etc., as transition triggers. However, it's not clear if 'error' is a generic placeholder for various error types or a specific event.

TABLE 12. Samples of incorrectly identified defects in Figures using GPT-4 model.

ID	Defect Info
FIG2.GPT4.4	The description mentions that both "PCV" and "PSV" can loop back on themselves, implying continuous operation. However, it doesn't specify what triggers the loop to continue or stop, other than the "stopVentilation" transition.
FIG3.GPT4.1	The description indicates that the "SelfTest" state stays in the same state if the event 'abortTest' or 'runSelfTest' occurs. However, it's ambiguous how the system should behave if these events occur simultaneously or in quick succession.

Figure 5 shows the distribution of severity ratings for all identified completeness defects. Most defects are considered to be of low importance, with 7 out of 9 falling into the lower severity categories of 1 or 2. However, there are two significant exceptions, classified as high-importance defects, each with a severity rating of 4. These two high-severity defects are associated with critical security and regulatory requirements in the software system, indicating a potential impact on compliance and safety.

D. AMBIGUITIES DETECTION ON FIGURES

In Table 11, key observations are noted regarding the correctly identified ambiguities by the GPT-4 model in various figures. The GPT-4 model demonstrates a capability to analyze and question transitions in state machine models effectively. For example, the models identify a missing transition in the 'Settings' mode that logically should lead to a 'powerOff' event. This transition is absent even though there are no explicit restrictions in the requirements prohibiting such an event (highlighted in FIG3.GPT4.2). Additionally, the LLM interprets various transitions in state diagrams that may introduce inconsistencies. In the case of FIG2.GPT4.2, the LLM recognizes a decision point in the UML diagram leading to two different paths to the 'StandBy' state, one of which lacks self-test requirements—a critical oversight in a medical system setting. The LLM also flags ambiguous

language within the diagrams, such as inconsistent labeling of transitional events, and suggests improvements for clarity (FIG4.GPT4.2).

However, in examining false positives, it is evident that GPT-4 often struggles with diagram syntax, misinterpreting transitions, and failing to grasp the diagrammatic representation as a human would. For instance, in Figure 2.1 of the SRS, the LLM incorrectly interprets transitions between two ventilation sub-events (PSV/PCV) as loop-back events and fails to recognize that two different triggering events are mutually exclusive, often resulting in redundant transitions (FIG3.GPT4.1). These false positives commonly arise due to the LLM's inability to handle missing information in the state machine and UML diagrams that are typically elaborated in the SRS, making it laborious to sift through and identify true positives.

Quantitatively, the GPT-4 model identified 7 true positives and 9 false positives, resulting in a precision rate of 0.44. Among the false positives, three were due to a lack of global contextual understanding, and another three were attributed to deficiencies in interpreting the syntax of UML diagrams. Regarding the impact of true positives, only one was rated as moderately severe (3/5), while the others were considered of minor importance to the system specifications (rated 1/5 or 2/5).

V. VERSIONING HISTORY AND DEFECTS

In addition to analyzing defects across three identified categories, we also investigated the detection of defects through the version control history of the SRS. The defects detected between the versions can be associated with any of the three types of defects that the LLM is prompted to identify. During the development of the final MLV specification document, participants at the ABZ conference highlighted specific issues that prompted the organizers to revise the SRS. This led to the creation of five versions (v1 to v5) of the document, each incorporating progressive modifications.

These modifications are detailed in the version history table at the beginning of the document. We utilized discussions from the GitHub issues page to understand the rationale behind each change made by the organizers in response to community feedback. Additionally, some requirements were modified independently of these community queries, primarily to simplify the modeling of the system. We employed GPT-4 LLM to trace these changes across each version of the document, analyzing versions v0 through v4. We focused on well-documented changes, searching for defects across the three previously discussed categories. Table 13 provides a summary of the defects identified in each version of the document, illustrating the evolution of the SRS through successive updates.

A. GPT-4 PERFORMANCE ON DEFECTS ACROSS VERSIONS

Table 14 illustrates the effectiveness of GPT-4 in detecting defects that have led to version updates in the document.

TABLE 13. Summary of version changes.

Version	Changes
v1	Ambiguity and inconsistency present in Figure 2 and FUN 10.5 regarding the flow
v2	Ambiguity in FUN 20 regarding the triggers of the breathing cycle
v3	Missing transition (completeness) in Figure 3.1. Ambiguity in Figure 3.1 regarding the condition to start ventilation state. Description of GUI 4 has ambiguity regarding the state machine transitions
v4	Ambiguity in Figure 3.1 regarding FailSafe state. Ambiguity (with partial incompleteness) in GUI 10 and GUI 50.2. The description of CONT 26 can be perceived ambiguous (or partially inconsistent)
v5	Ambiguity in GUI 1.2, GUI 1.3 and, GUI 5, regarding the state transition and setup. Similarly, Ambiguity (with partial inconsistency) in GUI 7 to GUI 9. Further, ambiguity (or completeness) errors in GUI 45.1 to GUI 45.3 regarding the Menu mode transition and loading of configurations from the saved file.

A top-down approach was utilized, where the earlier versions were cross-checked against all subsequent modifications. The GPT-4 exhibited varying degrees of success in identifying defects responsible for these version changes. Some defects spanned more than one defect category, such as Ambiguity and Completeness, and our analysis sought to identify LLM findings in either scenario. A notable observation from the table is that there are both straightforward and challenging examples where the LLM either consistently detected the defect across all versions or failed to detect it entirely. For instance, the issue of Ambiguity or Completeness in GUI 50.2 related to the self-test procedure was consistently detected across four versions. Conversely, ambiguities in GUI 1.2 and GUI 1.3 in version 5 remained undetected across all versions. Quantitatively, 22 defects were accurately identified by the LLMs (True Positives), and 27 were missed (False Negatives), resulting in a recall value of 0.45.

Certain versions of the SRS allowed the GPT-4 model to accurately identify defects, whereas others did not. For example, ambiguities in GUI 8 and GUI 9 in version 5 were only detected in version 2 of the SRS. Similarly, the ambiguity in 45.2 in version 5 was detected by all versions except version 1. These observations suggest that changes in the document's content significantly influence the LLM's ability to pinpoint specific defects, affected by the shifting contextual landscape. However, no clear trend was observed across versions regarding the detection of specific ambiguities; the detection seemed random and dependent on the specific version analyzed.

In cases where the GPT-4 model failed to detect changes across versions, two key observations were made. First, GPT-4 struggled to identify ambiguities in simple requirements such as GUI 1.2, which are clearly stated. Human experts identified these defects by cross-validating the requirements against other system components, particularly state machine diagrams. Second, the LLM was ineffective in identifying defects in figures that required an understanding of the system's state transitions. While human experts could

TABLE 14. Analysis of GPT-4 detection of defects in version control.

Version	Change	v0	v1	v2	v3	v4
v1	Amb. in Fig. 2.1	✓	-	-	-	
v2	Amb. in FUN 20	x	x	-	-	-
v3	Compl. in Fig. 3.1	x	x	x	-	-
v3	Amb. in GUI 4	x	x	✓	-	-
v4	Amb. in Fig. 3.1	✓	✓	✓	✓	-
v4	Amb./Compl. in GUI 10	x	✓	x	x	-
v4	Amb./Compl. in GUI 50.2	✓	✓	✓	✓	-
v4	Amb./Incon. in CONT 26	x	x	✓	✓	-
v5	Amb. in GUI 1.2 & GUI 1.3	x	x	x	x	x
v5	Amb. in GUI 5	x	✓	✓	x	x
v5	Amb/Incon. in GUI 7	✓	x	✓	x	x
v5	Amb/Incon. in GUI 8 & GUI 9	x	x	✓	x	x
v5	Amb/Compl. in GUI 45.2	✓	x	✓	✓	✓

identify missing transitions or erroneous flows depicted by the diagrams, the LLMs failed to recognize such errors, for instance, the completeness of Figure 3.1 in version 3.

On a positive note, GPT-4 has the potential to facilitate early defect detection, thus minimizing overall SRS defects from the outset. Various changes identified in versions 4 and 5 that were only recognized months after the document's initial release were detected by the LLM in earlier versions. This capability can enhance the document's quality, although some errors may persist.

VI. COMPARING GPT-4 IDENTIFIED DEFECTS AGAINST FORMAL MODELING

We have examined proceedings from the ABZ 2024 conference [16] to identify various defects highlighted by experts while formally modeling the MLV system. Our analysis involved five different research works on the MLV system, extracting defects present in the SRS document and categorizing them into three major categories. It is important to note that many defects identified often intersect between completeness and ambiguity. Table 15 displays the ambiguities identified by researchers applying different formal modeling approaches. When comparing defects identified by GPT-4, a detection is considered positive if GPT-4 identifies any related requirement with the same concern in its response. GPT-4 successfully recalls 2 out of the 5 ambiguities presented by experts in formal modeling.

Table 16 lists two inconsistencies identified by researchers. It is important to note that while these are labeled as inconsistencies, the modeling team does not strictly adhere to the formal definition of an inconsistency in requirements engineering. Consequently, we have also examined other defect categories in the GPT-4 results to ensure comprehensive matching. Both defects are addressed in the GPT-4 responses. However, while there is a discrepancy in the naming convention of figures, which lacks cross-referencing, we recognize the LLM's capacity to accurately identify issues in the language descriptions of figures.

Table 17 displays the incomplete requirements in the SRS as identified by various research works. We have permitted the GPT-4 results from the ambiguities to cross-reference these due to overlaps between the two categories.

TABLE 15. Ambiguities presented by human experts in ABZ 2024 conference.

Paper	Defect Info	Related Requirements	Detected by GPT-4
[55]	It is ambiguous whether the controller continuously checks the presence of undesirable events or not (e.g. communication with GUI)	CONT.13, CONT.14, CONT1.1.	x
[56]	In the modeling of the SelfTest step, there is no indication of how the controller is supposed to receive the results of the self-tests carried out by the controller and the GUI	FUN.6, CONT.17, CONT.18, CONT.19	✓
[57]	Subtle ambiguity in requirements where the powering off of the controller in PCV mode and subsequently powering it on can result in valves not being in safe mode	CONT.38	x
[57]	Alarm requirements need to be checked, e.g. SAV.16 states is not clear when the controller can be considered to be in a particular mode	SAV.16	x
[57]	It is not entirely clear whether the events depicted in the two State Machines of the MLV are intended to synchronize	CONT.38	✓

TABLE 16. Inconsistencies presented by human experts in ABZ 2024 conference.

Paper	Defect Info	Related Requirements	Detected by GPT-4
[58]	The language is not entirely consistent. Here, the mode that comes after the self-test has passed is called "Standby Mode" in the FUN requirements but is named "VentilationOff" in the CONT requirements	FIG.2.1 and FIG.4.1	✓
[58]	Lack of details could lead to an apparent conflict between CONT.24 and FUN22, regarding what mode will start at the end of the inspiration phase	CONT.24 and FUN.22	✓

TABLE 17. Completeness presented by human experts in ABZ 2024 conference.

Paper	Defect Info	Related Requirements	Detected by GPT-4
[59]	Missing timing requirement, which should specify what is the maximum latency between actual pressure dropping below target value and the controller making the decision that it is an inhalation attempt	FIG.1 and FIG.2	x
[56]	Lack of information, particularly at the level of the communication verification stage with the sensors on how long to wait before attempting a new connection	CONT.15	✓
[57]	SRS does not specify default values for some parameters (e.g., parameter RRAP); as every parameter needs to be initialized in mCRL2 modeling	PER.12	✓

Specifically, for the incomplete default parameter identified by [57], we consider any missing default parameter example in the GPT-4 results as a match. As shown, the GPT-4 model successfully recalls 2 out of the 3 identified incomplete requirements.

Overall, we found that 6 out of the 10 defects identified by human experts were also detected in the GPT-4 responses, yielding a recall value of 0.6. This indicates that preliminary analysis using the GPT-4 model can effectively assist requirements engineers in identifying defects in the SRS. However, while the human expert's analysis of each defect is more technically robust and directly linked to the system model, the GPT-4 responses tend to lack this level of precision and confidence.

VII. VALIDATION DATASET FOR REQUIREMENT ANALYSIS ASSISTANCE

In examining the requirements by stakeholders, including requirement engineers, previous research has indicated that automated assistance through a question-answer approach can significantly aid in analyzing NL requirements [22]. Early experiments demonstrated that LLMs such as ALBERT were useful in accurately answering questions from an SRS [22]. However, two key areas remain underexplored: (1) the potential of more advanced LLMs available today, and (2) the ability of LLMs to accurately retrieve questions related to system modeling. To address these gaps, we constructed a validation dataset for knowledge retrieval using the GPT-4 LLM model based on the MLV SRS. This dataset was created by scraping data from the GitHub Issues section of the MLV's official repository. The Issues section, used by ABZ conference participants to engage with the MLV's management team (comprising system experts), contains discussions focused on system functionalities. These discussions provide valuable insights for participants to model the system formally and verify its accuracy. By evaluating the effectiveness of GPT-4 in responding to participant inquiries, we aimed to assess their potential as technical assistants capable of resolving ambiguities about the system's behavior. The overall pipeline for generating and using the validation dataset for requirements analysis is presented in Figure 6.

In total, we retrieved 18 questions and their corresponding ground-truth answers. These ground-truth answers were supplied by the MLV system experts. We applied two approaches to feed context information to the GPT-4. First, we selected and provided only the context relevant to each specific inquiry. This approach involved feeding the GPT-4 model with only the required sections of the SRS before asking it to respond to the questions. Second, we supplied the LLM with the full context and then posed the same set of questions. The first approach is designed to be more focused, providing concise and pertinent information, while the second approach offers a global context. We expected the answers to be more accurate with the concise approach compared to the full-context approach. The prompt used in both scenarios to guide the GPT-4 model was: "The following contains the detailed software requirements…Assume you are the creator of the software requirements specification and need to answer queries about your requirements. Evaluate the content to answer the following question."

To evaluate the accuracy of the GPT-4 model responses, we compared each answer against its corresponding ground truth for validation. Additionally, we noted whether the answers included specific references to strengthen their explanations, a detail that human experts might overlook if relying solely on memory. We also assessed the verbosity of each response, as excessively long answers can be time-consuming to read and may lead to cognitive overload. This comprehensive analysis allowed us to gauge the accuracy of the GPT-4 model responses and their conciseness and ability to cite relevant information effectively.

A. GPT-4 KNOWLEDGE RETRIEVAL CAPABILITY

Table 18 presents a summary of the correct and incorrect responses generated by GPT-4 across various knowledge-retrieval datasets. The responses categorized as "Partially Correct" contain relevant information that can guide a human reader to deduce the answer, even if the LLM did not provide a direct response. These partially correct answers require additional time and effort to process. Our analysis suggests that LLMs excel at extracting explicit technical details from comprehensive Software Requirements Specification (SRS) documents. They can locate relevant information and refer to it in many cases, demonstrating a level of contextual understanding. GPT-4 can also leverage external

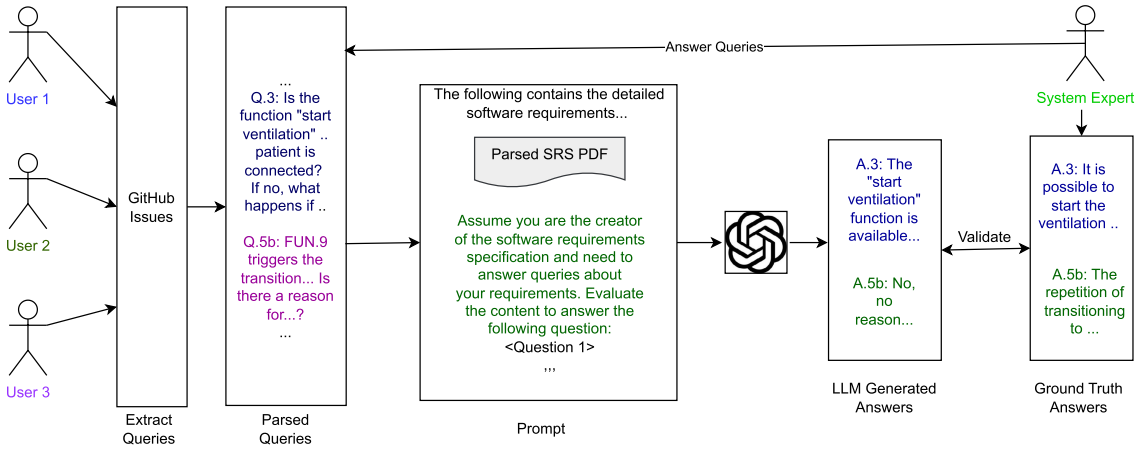


FIGURE 6. Pipeline for Analyzing GPT-4 Capabilities in Answering Questions to Aid Stakeholders in Ensuring the Quality of Requirements: Questions related to system modeling are extracted from GitHub issues and parsed to be fed to the GPT-4 model via the OpenAI API. System experts produce the Ground Truth (GT) answers to these questions. Each LLM-generated answer is then validated against the GT answers.

TABLE 18. Summary of GPT-4 responses to the knowledge-retrieval dataset.

	LLM with Relevant Context	LLM with Global Context
Correct	10	8
Partially Correct	5	5
Incorrect	3	5

TABLE 19. Performance analysis of the GPT-4 model on the knowledge-retrieval dataset.

	Ground Truth (Human)	LLM with Relevant Context	LLM with Global Context
Accuracy	-	0.77	0.62
Max Word Length	82	406	111
Avg Word Length	34	261	56
References Made	6	14	9

knowledge—similar to what a human expert might have about the system—to synthesize information from the SRS to generate technical answers. For instance, when evaluating the startup speed of the Controller subsystem relative to the GUI subsystem, the LLM could infer from figures and GUI requirements that the Controller might start before the GUI.

The results show that providing relevant context sections to GPT-4 yields more accurate answers than using a global context. This aligns with our hypothesis that focusing on specific sections leads to more accurate answers, as it reduces the LLM's exposure to extraneous information in lengthy technical documents. However, no consistent pattern was observed in the incorrect responses between the two data inputs. Many incorrect answers are associated with complex transitions requiring deeper analysis of flowcharts and technical domain-specific expertise. It is worth mentioning that even when LLMs provide incorrect answers, they often include references extracted from the SRS to support their response, indicating some level of contextual exploration, though the references may be misinterpreted.

Table 19 compares the GPT-4 answers with human-expert answers in terms of accuracy, verbosity, and references to specific locations within the SRS. Providing relevant context to the GPT-4 model increased its accuracy by 15% compared to using global context information. With relevant context, the LLM was more likely to cite specific requirements, resulting in better justification for its responses. However, human-expert answers referenced the SRS less frequently (only 6 out of 18 times), likely due to the expert's reliance on memory and intimate knowledge of the system. Despite fewer references, human-expert responses were more concise and direct compared to the LLM answers. On average, the GPT-4 model using relevant context contained 227 more words (a 668% increase), while the LLM with global context had 22 more words (a 65% increase). This increased verbosity indicates that while LLMs may offer more detailed explanations with references, they can also become cumbersome and overly wordy. Future research could focus on improving LLM prompts to encourage more concise responses, thereby aligning LLM behavior with the more focused approach typically used by human experts. This could result in LLM-generated answers that are more straightforward and easier to read without sacrificing accuracy or contextual relevance.

VIII. DISCUSSION

In this section, we discuss employing LLMs for requirements engineering tasks in-depth, illustrated through the MLV system case study. Our analysis focuses on the

efficacy of GPT-4 in requirements analysis, technical knowledge retrieval, and the detection of changes across versions.

LLMs offer substantial automation support, enhancing the efficiency of defect identification in software requirements. In our study, we evaluated the capability of the GPT-4 model to detect ambiguities, inconsistencies, and incompleteness in the industrial-level software specifications of the MLV system. We found that GPT-4 successfully identified several critical defects with variable precision across three defect categories. GPT-4 exhibited the highest precision in detecting completeness issues (average precision of 0.61), followed by inconsistencies (precision of 0.43), and ambiguities (precision of 0.39). Notably, five of the correctly identified defects were deemed significant to the MLV system.

However, it is crucial to recognize that LLMs are not a silver bullet for all RE challenges. Particularly in the identification of ambiguous and inconsistent requirements, the low precision highlighted the challenges in discerning valuable information presented by the LLM. While not a ''needle-in-a-haystack'' scenario, recognizing even moderate to minor issues can significantly enhance the software specification document. Nonetheless, careful scrutiny by engineers is required to filter out irrelevant and inaccurate 'false positive' defects.

We identified several characteristics of these false positives that underscore the limitations of GPT-4 in these tasks. Primarily, GPT-4 exhibits a constrained contextual window when analyzing requirements, which leads to numerous oversights and erroneous detections. Despite segment-specific analyses, GPT-4 struggles to integrate or utilize broader contextual information beyond their immediate focus, often resulting in various nonsensical false positives. This limitation is particularly detrimental to identifying consistency and ambiguity defects with high precision. We also show that providing the full context to an SRS does not help the LLM in connecting between the requirements, suggesting that the token limitation of an LLM model does not significantly affect their performance.

Moreover, the GPT-4 model demonstrated a consistently better performance in identifying defects across overall system requirements compared to specific components. In all defect categories, they achieved a precision greater than 0.5 with relatively fewer false positives, reflecting the general nature of their training data. However, the absence of nuanced domain understanding often leads to confusion over technical specifics that a human analyst would typically resolve.

We also leveraged versioning information from five different versions of the SRS to detect defects using GPT-4. Each version change was examined for defect identification across categories, and older versions were analyzed for early identification of later-corrected defects. We discovered that while some straightforward defects were consistently detected across all versions, more complex issues often went unnoticed. This observation underscores the limited contextual grasp of LLMs, evident in their inability to cross-validate with other requirements or comprehend state transitions in diagrams as effectively as a human expert might.

Finally, we compiled a knowledge-retrieval validation dataset to evaluate the LLMs' ability to respond to technical queries pertaining to the SRS. Correctly answering these questions can aid requirements engineers in analyzing requirement defects. These questions involved ambiguities, inconsistencies, and completeness issues critical for experts to accurately model the system and address any specification gaps or conflicts. The GPT-4 model was remarkably successful in generating comprehensive responses incorporating external knowledge, nearly mimicking human expert behavior. GPT-4 demonstrated a higher accuracy (77%) when provided with section-specific contexts compared to a global context (62%). This indicates that targeted information filtering in large document queries can significantly enhance LLM performance. However, it is worth noting that while LLMs tend to generate verbose responses, direct and concise answers are often more beneficial for users familiar with the system, although detailed explanations may be preferable in other contexts.

It is important to note that while LLMs show promise in identifying key issues within SRS written in natural language, they cannot fully replace the need for the later stages of formal system modeling, particularly in mission-critical software contexts. Formal modeling remains crucial for ensuring consistency in state transitions and the correctness of temporal components within a system. For instance, in our case study, issues pertaining to equations that impact the timely change of modes cannot be reliably flagged using current LLM capabilities. In mission-critical systems, ensuring the satisfaction of all guard conditions and invariants can be a life-or-death matter, underscoring the indispensable role of modeling experts in guaranteeing that the SRS document meets the requisite standards prior to the implementation stage. However, our findings also demonstrate that GPT-4 can identify several defects that experts have flagged following formal modeling. Overall, GPT-4 identifies a broader range of defects, providing a useful preliminary list that can help improve the quality of the SRS.

IX. CONCLUSION & FUTURE WORK

In this paper, we investigated the transformative potential of Large Language Models (LLMs) in the realm of requirements analysis, using a mechanical lung ventilator system as a case study. We demonstrated that GPT-4 can identify several defects in both the textual and graphical components of the Software Requirements Specification (SRS). Furthermore, GPT-4 can proactively detect potential future updates necessitated by these defects, often earlier than human reviewers. Additionally, LLMs assist in extracting technical knowledge from the SRS, facilitating the formal modeling of the system. However, it is critical to acknowledge that LLMs

are not a panacea for all challenges in requirements analysis. Several limitations compromise their efficacy, including a constrained contextual window, a deficiency in specific domain expertise, and an overall lack of nuanced human-like understanding of software contexts.

Looking ahead, our future work will explore the application of few-shot learning to enhance the defect-detection capabilities of LLMs. By training LLMs with select examples from relevant or analogous projects, we aim to refine their precision in pinpointing requirement defects. Additionally, while our current methodology employs straightforward and intuitive prompt designs, we are interested in experimenting with more complex strategies, such as chain-of-thought (CoT) prompting, to potentially elevate the role and performance of LLMs. We plan to extend our research to additional case studies, which will help in formulating a more generalized understanding of LLMs' performance across diverse industrial software requirements documents. Moreover, the exploration of the 'temperature' parameter in LLM settings remains an open avenue for future research, which will allow us to assess the stability and predictive quality of LLM outputs when configured to take more calculated risks.

Future research can also explore Retrieval Augmented Generation (RAG) in requirements engineering. RAG automatically gathers and synthesizes information from extensive software documentation, user manuals, technical specifications, and industry best practices [60]. This approach has been used in requirement engineering tasks such as test scenario generation [61]. The retrieval component searches a vast repository for pertinent data, while the generation component uses this data to produce detailed, accurate, and contextually rich analyses, recommendations, or summaries [60]. RAG enhances LLM capabilities by providing up-to-date external information and improving the detection of ambiguities, inconsistencies, and incompleteness. For example, it can cross-reference requirements to ensure alignment, flag contradictions, and compare requirements with comprehensive templates to identify missing elements. Leveraging RAG, LLM-based systems can offer detailed, context-aware analyses, enhancing the accuracy and reliability of software requirement evaluations.

REFERENCES

[1] G. Sandhu and S. Sikka, "State-of-art practices to detect inconsistencies and ambiguities from software requirements," in *Proc. Int. Conf. Comput., Commun. Autom.*, May 2015, pp. 812–817.

[2] A. A. Alshazly, A. M. Elfatatry, and M. S. Abougabal, "Detecting defects in software requirements specification," *Alexandria Eng. J.*, vol. 53, no. 3, pp. 513–527, Sep. 2014.

[3] M. Jackson, *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. Reading, MA, USA: Addison-Wesley, 1995.

[4] A. Aurum and C. Wohlin, *Engineering and Managing Software Requirements*, vol. 1. Cham, Switzerland: Springer, 2005.

[5] R. Sharma and K. Biswas, "Resolving inconsistency and incompleteness issues in software requirements," *Manag. requirements Knowl.*, vol. 4, pp. 245–263, Sep. 2013.

[6] A. K. Thurimella and D. Janzen, "Metadoc feature modeler: A plug-in for IBM rational DOORS," in *Proc. 15th Int. Softw. Product Line Conf.*, Aug. 2011, pp. 313–322.

[7] W. X. Zhao et al., "A survey of large language models," 2023, *arXiv:2303.18223*.

[8] C. Urban and A. Miné, "A review of formal methods applied to machine learning," 2021, *arXiv:2104.02466*.

[9] I. Atoum, M. K. Baklizi, I. Alsmadi, A. A. Otoom, T. Alhersh, J. Ababneh, J. Almalki, and S. M. Alshahrani, "Challenges of software requirements quality assurance and validation: A systematic literature review," *IEEE Access*, vol. 9, pp. 137613–137634, 2021.

[10] P. Cousot, "Verification by abstract interpretation," in *Verification: Theory and Practice: Essays Dedicated To Zohar Manna on the Occasion of His 64th Birthday*. Springer, 2003, pp. 243–268.

[11] P. Pudlák, "The lengths of proofs," in *Studies in Logic and the Foundations of Mathematics*. Amsterdam, The Netherlands: Elsevier, 1998, pp. 547–637.

[12] R. Jhala and R. Majumdar, "Software model checking," *ACM Comput. Surveys (CSUR)*, vol. 41, no. 4, pp. 1–54, 2009.

[13] A. Fantechi, S. Gnesi, and L. Semini, "Rule-based NLP vs ChatGPT in ambiguity detection, a preliminary study," in *Proc. REFSQ Workshops*, 2023, pp. 1–26.

[14] C. Arora, J. Grundy, and M. Abdelrazek, "Advancing requirements engineering through generative AI: Assessing the role of LLMs," 2023, *arXiv:2310.13976*.

[15] D. Luitel, S. Hassani, and M. Sabetzadeh, "Using language models for enhancing the completeness of natural-language requirements," in *Proc. Int. Work. Conf. Requirements Eng. Found. for Softw. Qual.*, 2023, pp. 87–104.

[16] S. Bonfanti and A. Gargantini, "The mechanical lung ventilator case study," in *Proc. Int. Conf. Rigorous State-Based Methods*, 2024, pp. 281–288.

[17] C. Masmoudi, P. Marange, E. Bonjour, E. Levrat, and A. Kerbrat, "Adopting formal methods on requirements verification and validation for cyber-physical systems: A systematic literature review," *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 3274–3279, 2022.

[18] M. Krichen, "Formal methods and validation techniques for ensuring automotive systems security," *Information*, vol. 14, no. 12, p. 666, Dec. 2023.

[19] K. Mokos, T. Nestoridis, P. Katsaros, and N. Bassiliades, "Semantic modeling and analysis of natural language system requirements," *IEEE Access*, vol. 10, pp. 84094–84119, 2022.

[20] L. Belzner, T. Gabor, and M. Wirsing, "Large language model assisted software engineering: Prospects, challenges, and a case study," in *Proc. Int. Conf. Bridging Gap Between AI Reality*, 2023, pp. 355–374.

[21] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "Automated handling of anaphoric ambiguity in requirements: A multi-solution study," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, May 2022, pp. 187–199.

[22] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "AI-based question answering assistance for analyzing natural-language requirements," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, May 2023, pp. 1277–1289.

[23] 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," *ACM Comput. Surveys*, vol. 54, no. 3, pp. 1–41, Apr. 2022.

[24] N. Marques, R. R. Silva, and J. Bernardino, "Using ChatGPT in software requirements engineering: A comprehensive review," *Future Internet*, vol. 16, no. 6, p. 180, May 2024.

[25] V. Castañeda, L. Ballejos, M. L. Caliusco, and M. R. Galli, "The use of ontologies in requirements engineering," *Global J. researches Eng.*, vol. 10, no. 6, pp. 2–8, 2010.

[26] A. Handbook, *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity*, 2003.

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

[28] *IEEE Recommended Practice for Software Requirements Specifications*, Standard IEEE STD-830, 1998.

[29] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2023, *arXiv:2307.06435*.

[30] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[31] V. Ashish, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–20.

[32] M. Zhou, N. Duan, S. Liu, and H.-Y. Shum, "Progress in neural NLP: Modeling, learning, and reasoning," *Engineering*, vol. 6, no. 3, pp. 275–290, Mar. 2020.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[34] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," Tech. Rep., 2018.

[35] Q. Wei, Z. Yao, Y. Cui, B. Wei, Z. Jin, and X. Xu, "Evaluation of ChatGPT-generated medical responses: A systematic review and meta-analysis," *J. Biomed. Informat.*, vol. 151, Mar. 2024, Art. no. 104620.

[36] Q. Li, L. Fu, W. Zhang, X. Chen, J. Yu, W. Xia, W. Zhang, R. Tang, and Y. Yu, "Adapting large language models for education: Foundational capabilities, potentials, and challenges," 2023, *arXiv:2401.08664*.

[37] H. Zhao, Z. Liu, Z. Wu, Y. Li, T. Yang, P. Shu, S. Xu, H. Dai, L. Zhao, G. Mai, N. Liu, and T. Liu, "Revolutionizing finance with LLMs: An overview of applications and insights," 2024, *arXiv:2401.11641*.

[38] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali, and S. Azam, "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26839–26874, 2024.

[39] W. Chen, "Large language models are few(1)-shot table reasoners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 22199–22213.

[40] A. Chhabra, H. Askari, and P. Mohapatra, "Revisiting zero-shot abstractive summarization in the era of large language models from the perspective of position bias," 2024, *arXiv:2401.01989*.

[41] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–37, Mar. 2019.

[42] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Comput. Surveys*, vol. 53, no. 3, pp. 1–34, May 2021.

[43] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Qian Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," 2024, *arXiv:2403.14608*.

[44] Z. Wang, B. Bi, S. K. Pentyala, K. Ramnath, S. Chaudhuri, S. Mehrotra, Z. Zixu, X.-B. Mao, and S. Asur, "A comprehensive survey of LLM alignment techniques: RLHF, RLAIF, PPO, DPO and more," 2024, *arXiv:2407.16216*.

[45] T. R. McIntosh, T. Susnjak, T. Liu, P. Watters, and M. N. Halgamuge, "The inadequacy of reinforcement learning from human feedback—Radicalizing large language models via semantic vulnerabilities," *IEEE Trans. Cognit. Develop. Syst.*, vol. 16, no. 4, pp. 1561–1574, Aug. 2024.

[46] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 2339–2356.

[47] M. Chen, G. He, and J. Wu, "ZDDR: A zero-shot defender for adversarial samples detection and restoration," *IEEE Access*, vol. 12, pp. 39081–39094, 2024.

[48] OpenAI et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[49] (2024). *Hello GPT-4O*. Accessed: May 29, 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[50] J. Manyika and S. Hsiao. (2023). *An Overview of Bard: An Early Experiment With Generative AI*. [Online]. Available: https://ai.google/static/documents/google-about-bard.pdf

[51] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.

[52] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2024, *arXiv:2402.06196*.

[53] R. Anil et al., "PaLM 2 technical report," 2023, *arXiv:2305.10403*.

[54] A. Hajikhani and C. Cole, "A critical review of large language models: Sensitivity, bias, and the path toward specialized AI," *Quant. Sci. Stud.*, vol. 1, pp. 1–21, Jul. 2024.

[55] A. Mammar, "An event-B model of a mechanical lung ventilator," in *Proc. Int. Conf. Rigorous State-Based Methods*, 2024, pp. 307–323.

[56] A. R. Ndouna and M. Frappier, "Modelling the mechanical lung ventilation system using TASTD," in *Proc. Rigorous State-Based Methods 10th Int. Conf.*, 2024, pp. 1–20.

[57] D. van Dortmont, J. J. Keiren, and T. A. Willemse, "Modelling and analysing a mechanical lung ventilator in mcrl2," in *Proc. Int. Conf. Rigorous State-Based Methods*, 2024, pp. 341–359.

[58] M. Farrell, M. Luckcuck, R. Monahan, C. Reynolds, and O. Sheridan, "Fretting and formal modelling: A mechanical lung ventilator," in *Proc. Int. Conf. Rigorous State-Based Methods*, 2024, pp. 360–383.

[59] P. Tokariev and F. Mallet, "Real-time CSSL: Application to the mechanical lung ventilator," in *Proc. Int. Conf. Rigorous State-Based Methods*, 2024, pp. 289–306.

[60] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on RAG meeting LLMs: Towards retrieval-augmented large language models," 2024, *arXiv:2405.06211*.

[61] C. Arora, T. Herda, and V. Homm, "Generating test scenarios from NL requirements using retrieval-augmented LLMs: An industrial study," 2024, *arXiv:2404.12772*.

TASLIM MAHBUB (Member, IEEE) received the B.S. degree (Hons.) in computer engineering from American University of Sharjah, United Arab Emirates, in 2021, and the M.S. degree in computer science, specializing in artificial intelligence from Khalifa University, Abu Dhabi, United Arab Emirates, in 2023. He is currently a Research Associate with the Department of Computer Science and Engineering, American University of Sharjah. His research interests include machine learning, deep learning, and practical AI software systems.

DANA DGHAYM received the Ph.D. degree in computer science from the University of Southampton, in 2017. Following the Ph.D. degree, she was a Postdoctoral Researcher and then a Lecturer with the Cyber-Physical Systems Group, School of Electronics and Computer Science, University of Southampton. She is currently an Assistant Professor with the Department of Computer Science and Engineering, American University of Sharjah. Her main research interests include formal methods for software engineering, with a specialization in model-based formal methods, particularly Event-B. She has contributed to the Event-B structured refinement and composition methodologies and has experience in tool development and verification in aerospace, railway, and maritime autonomous systems. She has participated in various European and U.K. projects aimed at facilitating the industrial use of formal methods.

AADHITH SHANKARNARAYANAN (Member, IEEE) is currently pursuing the Bachelor of Science degree in computer science with American University of Sharjah. His research interests include computer science, such as natural language processing and computer vision with a focus on object detection and segmentation, underwater image restoration, and machine learning, particularly in feature selection, extraction, and prediction.

TAUFIQ SYED is currently pursuing the Bachelor of Science degree in computer science with American University of Sharjah. His research interests include large language models and generative AI, where he is passionate about exploring the capabilities and applications of these advanced technologies.

SALSABEEL SHAPSOUGH received the B.S. and M.S. degrees in computer engineering from American University of Sharjah, United Arab Emirates, in 2015 and 2017, respectively. She is currently a Laboratory Instructor with the Department of Computer Science and Engineering, American University of Sharjah. Her teaching areas include programming, software design, and computer architecture and organization. Her research interests include the Internet of Things, smart solar energy, smart education, machine learning, and deep learning.

IMRAN ZUALKERNAN (Member, IEEE) received the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 1991.

He is currently a Professor and the Department Head of the Department of Computer Science and Engineering, American University of Sharjah. He has taught with the University of Minnesota and Pennsylvania State University and has held C-level positions in various startup companies. He has published more than 80 papers in refereed conference proceedings and journals. His teaching areas include advanced learning technology, low-cost hardware/software co-design, and the Internet of Things. His current research interests include ubiquitous tangible learning systems, wearable learning, adoption and alignment models for learning technologies, advanced learning technologies, the Internet of Things, and software design.