Using a Local LLM to Inspect Security Requirements of IoT Software Systems

Christian Noronha Moreira
Escola Politécnica de Engenharia, Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
christian.moreira@poli.ufrj.br

Bruno Pedraça de Souza
PESC/COPPE, Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
bpsouza@cos.ufrj.br

Guilherme Horta Travassos
PESC/COPPE, Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
ght@cos.ufrj.br

Abstract

The software development life cycle involves creating artifacts that describe a system's specification. One of these artifacts is the requirements document. Identifying potential defects in these artifacts is essential to preventing defects in these systems and mitigating them throughout the development cycle. Thus, software inspections are widely used as quality assurance methods used to detect defects in such artifacts. On one hand, software inspections identify up to 60% of defects in artifacts. On the other hand, they require considerable time and human effort if not properly planned and performed. The objective of this work is to present a solution based on local Large Language Models (LLMs) for supporting the inspection of software requirements. A LLM-based workflow was developed by applying local models, prompt engineering techniques, and a previous checklist-based inspection technique focused on security requirements in Internet of Things (IoT) software systems. To evaluate the proposed solution, we performed an observational study to compare the results of an inspection session of a collection of requirements artifacts using the local LLM Workflow with the results of previous inspection sessions performed by novice inspectors on the same artifacts and using the same inspection technique. The emerging results showed limited effectiveness when inspecting all functional and security requirements jointly. However, the local LLM-based workflow achieved a promising effectiveness of over 70% when focused on security-related defects, the object of interest of the target checklist.

Keywords

Requirements inspection, LLM, local models, prompt engineering, security requirements, Internet of Things

1 Introduction

The Internet of Things (IoT) integrates software, devices, gateways, and peopleware, creating increasingly complex software systems that require special attention to security throughout the development life cycle [2, 9, 11]. Due to the large number of interconnected components and the sensitivity of exchanged data, identifying security defects as early as possible is essential.

Software inspections are a well-established Software Quality Assurance (SQA) technique for detecting defects during early development stages, reducing the cost of fixing problems in later phases [14, 18]. However, inspections are often expensive in terms of time and human effort, and their effectiveness depends on factors such as the inspection technique, inspector expertise, and artifact complexity [5].

Large Language Models (LLMs) have recently shown promise in supporting software engineering activities involving textual artifacts, including requirements inspection [1, 23]. Nevertheless, cloud-based LLMs raise confidentiality concerns because software projects frequently involve sensitive information [22]. Local LLMs mitigate these concerns by keeping data within the organization, although they may provide lower performance than larger cloud-based models [12, 13, 24]. Prompt engineering[1] can help overcome these limitations by providing structured instructions and contextual information [20].

This paper presents a local LLM-based workflow to support the inspection of IoT security requirements written in Brazilian Portuguese. The workflow combines prompt engineering techniques with IoTSecIT [9], a checklist specifically designed to inspect functional and security requirements of IoT software systems. The proposed workflow was evaluated by inspecting real IoT requirements artifacts and comparing the obtained results with a reference baseline derived from previous human inspections of the same artifacts. The results provide initial indication that local LLMs can support requirements inspections with effectiveness comparable to human inspectors. Thus, the main contributions of this paper are:

- A generic workflow that can be tailored to any semi-automated inspection process using checklists.
- Indications of the feasibility of using local LLMs as supporting mechanisms for software inspections, exemplified with security defects.
- An exploratory evaluation of prompt engineering techniques applied to software requirements inspection in Brazilian Portuguese.
- An initial empirical comparison between human inspectors and a local LLM-based inspection in the context of IoT software systems.

Apart from this introduction, this paper is organized as follows. Section 2 presents the related work. Section 3 demonstrates the conception of the LLM-based workflow. Section 4 details the observational study performed to evaluate our proposed workflow. Section 5 reports the main findings of the evaluations, including the discussion and limitations. Finally, Section 6 concludes the paper and outlines future research directions.

[1] Prompt engineering is the process of designing prompts that guide LLMs toward generating more accurate and reliable outputs [6].

2 Related Work

There are relatively few studies addressing the application of LLMs for performing inspections on software development lifecycle artifacts. Although the technical literature contains numerous studies on the use of LLMs in software development, most are primarily focused on requirements engineering, coding, and software testing activities.

Mahbub et al. [28] investigated the capability of a large-scale LLM, containing over one trillion parameters, to identify ambiguity, inconsistency, and incompleteness defects in software requirements. The study applied zero-shot prompting enhanced with contextual information using an IoT software system requirements specification as the artifact for the study. Each document section was analyzed individually, with dedicated prompt templates designed for each defect category. The findings demonstrated that detection accuracy improved when global contextual information was incorporated into the prompts, achieving the best results when the context was specifically tailored to both the document section and the inspection task.

Biswas and Das [27] introduced ARIA-QA, an AI-based agentic framework designed to support requirements analysis through a question-and-answer approach. The solution employs two LLM-based agents enhanced with Retrieval-Augmented Generation (RAG), named "Verifier" and "Analyzer," which iteratively interact by generating questions, producing answers, and suggesting requirement modifications to address identified defects. This process resembles checklist-based inspection, although it relies on dynamically generated questions rather than predefined static checklists. The study concluded that ARIA-QA demonstrated promising effectiveness in identifying inconsistency and incompleteness defects within software requirements.

Uygun et al. [29] proposed a solution utilizing five local LLMs, ranging from 7 to 13 billion parameters, for the analysis of system requirements in the automotive domain. The approach incorporated Retrieval-Augmented Generation (RAG) supported by a dataset of 10 documents containing 1,640 requirements. For evaluation purposes, the authors defined seven assessment questions to measure system performance. The results indicated that the proposed solution was capable of processing complex queries, extracting relevant information from the requirements documents, and generating satisfactory responses according to the qualitative evaluation conducted by the authors. Among the evaluated models, the 13-billion-parameter variants demonstrated superior performance.

The studies presented in this section show that LLMs are good alternatives for automating requirements analysis tasks, which highlights the importance of additional prompt engineering techniques to achieve satisfactory results. Unlike the related studies presented, our study focuses on applying local LLMs to evaluate the security requirements of IoT software systems. To our knowledge, we have not found any study that has used this configuration.

3 Design of an Inspection Workflow Using a Local LLM

This section introduces the workflow built to perform inspections on software requirements specification documents, illustrated in Figure 1. The solution receives a document in text or PDF format and uses configuration files to define the chosen model, execution parameters, and prompting strategies. The workflow architecture allows the execution of different inspection techniques simply by editing the prompts and parameters present in the configuration files. To execute local LLMs, the chosen solution was Ollama [17], which offers a catalog of different models in various scales.

The workflow is composed of two main components: the requirements treatment component and the inspection component.

3.1 Requirements treatment component

In the requirements processing stage, the documents are converted into text format when necessary. Then, specific keys from the configurations will define whether the document needs to be processed, which means the separation of the general project description from the list of requirements. As Figure 1 illustrates, there are two possible paths when the document is formatted to text.

- **The input document was not pre-treated**: In that case, the requirements treatment component will instruct the configured model to generate the list of requirements and the project description. The instruction is a prompt registered in the prompt files folder, and the chosen prompt is defined by the configuration file.
- **The input document was pre-treated**: With the pre-treated document, the model is no longer necessary, and the text content is forwarded as-is.

This step is essential to standardize the entry structure for the subsequent stages of the workflow. After this processing phase, the requirements are forwarded to the inspection module.

3.2 Inspection component

The inspection component was designed to incorporate system prompts for contextualization and role assignment, as well as separate example-based prompts simulating turns between the user and the assistant, followed by the final instruction responsible for guiding the inspection task. This structure enables the application of the previously discussed prompt engineering techniques. System prompts support the adoption of role-based prompting [8], while the interactions enable the use of few-shot prompting [8][16][4] and chain-of-thought prompting [8] strategies. Beyond that, by enabling the execution of multiple inspections across different iterations, the component also supports the application of the self-consistency strategy [21]. Figure 2 illustrates an excerpt of the system prompt used for instructing the inspection, whereas Figure 3 presents an excerpt of the user and the assistant chat turns added to the prompt that serve as few-shot examples. The content of Figures 2 and 3 is in Brazilian Portuguese to keep fidelity with the implementation.

This component is responsible for executing analyses based on the configured instructions. Each model interaction employs the configured prompts, while additional parameters define whether the execution will adopt a self-consistency strategy and how many interactions will be performed in such cases. A single inspection produces the final list of identified discrepancies[2] if executed without self-consistency. Otherwise, the component performs n inspections,

[2]A discrepancy refers to any inconsistency, or contradiction identified during the inspection of a requirements specification artifact. After further analysis, a discrepancy may be classified either as a real defect or as a false positive.

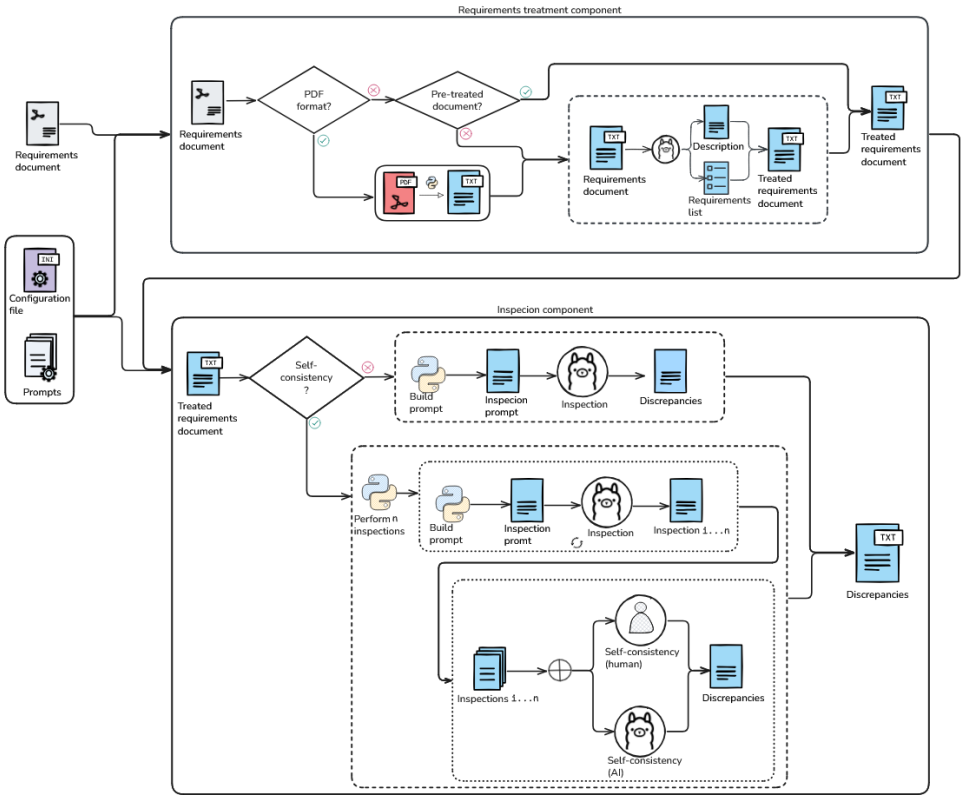

Figure 1: Proposed workflow

```
<|im_start|>system
Você é um especialista em Qualidade de Software e Engenharia de Requisitos. Você receberá um texto com
a descrição de um sistema de software, e uma lista de requisitos.
Esse texto é um consolidado de artefatos do software. Baseado nesse documento, você deve inspecioná-lo,
ou seja, encontrar possíveis defeitos nos requisitos.
Considere os tipos de defeito, e suas descrições:

- Omissão: Informação necessária sobre o software omitida no artefato.
- Fato incorreto: Informação no artefato do software contradiz informação do documento de requisitos ou
conhecimento geral.
- Inconsistência: Informação de uma parte do artefato do software é inconsistente com outra parte.
- Ambiguidade: Informação no artefato permite diferentes interpretações.
- Informação estranha: Informação não é necessária ou usada.
- Defeito vindo da técnica aplicada (exemplo: checklist)
Apenas retorne os defeitos, e a explicação do porquê, sem o tipo de defeito.
Caso haja dúvida, se um requisito "talvez" seja um defeito ou não, considerar como defeito.

Além disso, a inspeção deve ser baseada em checklist. Neste tipo de inspeção, será enviada também uma
lista de considerações usadas na inspeção. A análise da checklist é feita em paralelo com a análise dos
requisitos, e as perguntas podem ajudar a encontrar defeitos.
Perguntas relacionadas a requisitos gerais:
 Pergunta 1: O tipo de dado coletado é especificado? (Temperatura, umidade, poluição, etc.)
 Pergunta 2: Quem ou o que coleta os dados é especificado? (Sensores, leitores de QR code, etc.)
 Pergunta 3: Quem ou o que gerencia os dados coletados é especificado? (Administrador, tomador de
```

Figure 2: Excerpt of used prompt

```
  [SEG04] No momento do cadastro, o usuário deve escolher senhas com caracteres especiais, letras e
números.
  [SEG05] O sistema deve obrigar a atualização das senhas a cada 3 meses.
[FIM REQUISITOS E DESCRIÇÕES]

<|im_end|><|im_start|>assistant
Vamos analisar requisito por requisito.
 [RF01] O requisito introduz o sensor, a medida (umidade do solo) e unidade (porcentagem, %%). Porém,
15 segundos é um tempo muito baixo para ocorrer uma alteração relevante na umidade do solo. Isso
configura um defeito de Fato Incorreto.
 [RF02] O limite de umidade configurado é citado no requisito [RF03]. Sem defeitos.
 [RF03] O requisito introduz o cronograma de irrigação e o limite de umidade como configurações, usadas
em outros requisitos. Sem defeitos encontrados.
 [RF04] A configuração do limite de consumo não é citada por nenhum outro requisito. Isso configura um
defeito de Omissão.
 [RF05] O requisito introduz os atuadores cadastrados (cadastro de dispositivos introduzido no
requisito [RF10]) e cita a configuração de cronograma introduzida no requisito [RF03]. Sem defeitos
encontrados.
 [RF06] Sem defeitos.
 [RF07] Grupos de acesso introduzidos no [RF06]. Sem defeitos.
 [RF08] Sem defeitos.
 [RF09] Sem defeitos.
 [RF10] Introduz o cadastro de dispositivos no sistema, conceito usado em outros requisitos. Sem
defeitos encontrados.
```

Figure 3: Excerpt of few-shot examples

generating ***n*** independent results (referred to as samples), which are consolidated by the model in order to return the final discrepancy list. In addition, each sample is exported individually, allowing a human inspector to analyze and consolidate the results manually.

4 Observational Study

In order to evaluate the proposed local LLM-based workflow, it was configured to perform the IoTSecIT [9] together with the presented prompting techniques. Table 1 presents some of IoTSecIT's security-related questions that the LLM-based workflow uses. Unlike free-form prompting, a checklist-based inspection technique provides an explicit reasoning structure for the LLM. Each checklist item represents a specific verification criterion that guides the model to evaluate particular aspects of the requirements specification. Consequently, IoTSecIT was considered suitable for integration with local LLMs because it constrains the inspection process, reduces ambiguity, and promotes a more systematic analysis of functional and security requirements.

This observational study aims to verify if the LLM-based workflow is feasible, i.e, it can detect software (including security) defects comparable to human inspectors when carrying out a software requirements specification inspection. Therefore, it is conjectured

Table 1: Excerpt of De Souza's Checklist 2025

Nº	Questions - Security defects
1	Are security requirements identified?
2	Were the roles defined according to the information that each system user can access?
3	Are passwords required to be updated every x days?
4	Are encrypted communication channels required?
5	Is there a need to have backup policies?
6	Is it specified that only devices registered in the system send and receive data?

that the proposed LLM-based tool is feasible for detecting defects in functional and security requirements of IoT software systems. Then, the main research question of this study is: *"Can a local LLM-based workflow perform functional and security-focused software requirements inspections for IoT software systems with performance similar to human inspectors?"*

Then, based on GQM [3], the observational study aims to **analyze** the proposed LLM-based workflow **with the purpose of** characterizing it **in relation to** its effectiveness and similarity of detecting defects when compared to humans **from the point of view of** software engineering researchers **in the context of** evaluating functional and security defects in actual IoT software requirements artifacts.

Instrumentation: For this study, we used some instruments to perform the semi-automated inspection, such as IoT software requirements artifacts from a real system, IoTSecIT, and the defect report form. The requirements artifacts contain information about the problem domain, functional, and non-functional requirements of an actual IoT software system regarding the monitoring of biosecurity conditions of installations written in the Portuguese Language.

Study Context: The SAFE Project is an IoT-based software system developed to monitor biosecurity conditions in shared environments, such as laboratories, screening centers, and waiting rooms, through real-time data collected with IoT devices [19]. These devices monitor the number of people in the environment, CO2 levels, humidity, and temperature, while the system provides interactive visualization, risk alerts, and maintenance management services. The SAFE consists of four main subsystems: BIoTs, responsible for environmental data collection; Manager, which handles user and facility management; Broker, which supports subsystem communication via MQTT; and Dashboard, which visualizes system data [10]. For this study, requirements specification artifacts from the three core subsystems (BIoT, Manager, and Dashboard) were selected as inspection targets due to their central role in the software system. The used middleware (RabbitMQ) is out of the scope of our study. Figure 4 presents an excerpt of the project's requirements.

Pilot: We performed a pilot study to verify and calibrate the prompts used in the LLM-based workflow. IoT requirements artifacts from another IoT project (not the same project used in the observational study) were used. After that, the data was collected and analyzed to determine if the observational study protocol was adequate and performable.

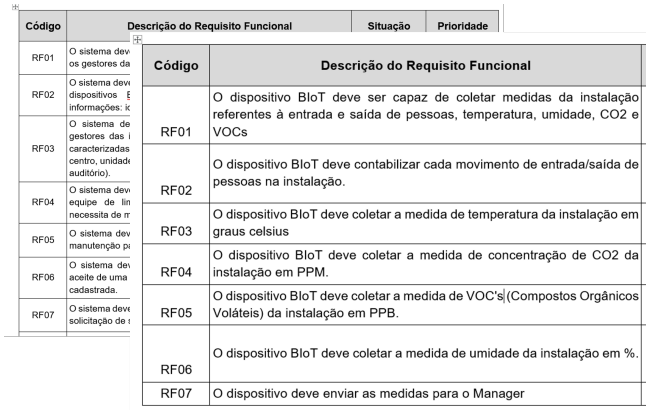

Figure 4: Excerpt of SAFE Project's artifacts

Variables: LLM configuration and prompts were considered as independent variables. The dependent variables were the number of defects and effectiveness.

Measurements: The measures used in the study protocol to support the observation of the dependent variables are:

- **Number of detected defects**: The sum of all distinct defects identified in the artifacts throughout the inspection sessions.
- **Effectiveness**: the ratio between the number of defects detected per inspector (humans or LLM) divided by the number of known defects in the artifact

Experimental Setup: The workflow was set up following the description explained below:

- **Processing**: Core i7, 32 GB of RAM, plus an RTX 4060 8 GB GPU.
- **Local model**: Qwen3 14b [15] run in Ollama [17].
- **Context window size**: 32,000 tokens.
- **Temperature**: 0.3.
- **Model layers running on GPU**: 18 layers.
- **Number of self-consistency executions (*n*)**: 3 executions.

Baseline: To establish a basis for comparison, a reference oracle was created. This oracle was derived from previously conducted inspection studies involving human participants, who inspected the same artifacts using the same IoTSecIT inspection checklist (these studies were not the focus of the present work and were used exclusively to construct the evaluation baseline). After the completion of these studies, all reported discrepancies were consolidated and independently assessed by three researchers with expertise in software quality. Each discrepancy was classified as either a valid defect or a false positive. The validated defects were then consolidated into a reference baseline, which served as the oracle for evaluating the effectiveness of the proposed LLM-based workflow. It is worth noting that, before participating in the main inspection sessions, the novice inspectors had taken part in previous inspection assignments to gain experience with the inspection process and the checklist. Table 2 summarizes the resulting baseline, showing the total number of known defects identified for each artifact (Manager, BIoT, and Dashboard). Security defects are included in these totals. The final baseline comprises 46 known defects for

the Manager artifact, including eight security defects; 39 known defects for BIoT, including six security defects; and 30 known defects for the Dashboard artifact, none of which are security-related.

Table 2: Baseline built from the inspection sessions

Subsystem	Total Defects	Security-related defects
Manager	46	8
BIoT	39	6
Dashboard	30	not applied
Total	115	14

Study Execution: The study was conducted in the second half of 2025 to evaluate IoT requirements artifacts using local (offline) LLMs combined with prompt engineering techniques. The study was carried out by three researchers: an undergraduate researcher with three years of experience in software development, a doctoral researcher with approximately eight years of experience in software testing and inspection, and a senior software engineering researcher. As it has been said, the study aimed to evaluate the feasibility of using a local LLM to identify defects in software requirements artifacts, particularly in comparison with the defects identified by novice software engineers. For this, we used prompting techniques such as role-based prompting, few-shot prompting, chain-of-thought prompting, and self-consistency to instruct the LLMs to apply a checklist-based inspection technique, in this case, the IoTSecIT [9].

5 Results, Discussion, and Limitations

5.1 Results

The results were evaluated from two perspectives: all defects found (functional and security-based), and only those focused on security. Table 3 shows the comparative results of the inspection sessions with humans (identified as IS1, IS2, and IS3) against the oracle results. The inspection sessions IS1, IS2, and IS3 were conducted in 2024 and 2025 with the purpose of evaluating the effectiveness, efficiency, and cost-effectiveness of the proposed checklist. In addition, these inspection sessions were conducted with undergraduate students who we consider to be novices [7]. Thus, the baseline was constructed as the sum of all distinct defects identified during these inspection sessions.

Table 3: Comparison of the number of defects identified by each inspection session in each artifact

Inspection Session	Manager	BIoT	Dashboard
IS1	23	28	22
IS2	16	11	6
IS3	18	14	12
LLM tool	11	11	14

We created a comparative table of studies to present the effectiveness of the LLM tool in identifying defects in individual artifacts. Table 4 presents a comparison of the effectiveness of the LLM tool technique with previous inspection sessions. To assess the effectiveness of the inspection carried out by the LLM-based tool, we divided the number of defects identified by each inspection session in each artifact by the total number of known defects in that artifact. For example, the inspection session with the LLM-based tool identified 11 defects in the Manager artifact; thus, according to the baseline, these 11 identified defects were divided by 46 (known defects in the Manager in Table 2), totaling an effectiveness of 23.91%.

Table 4: Comparative effectiveness of inspection sessions

Inspection Session	Eff. in Manager	Eff. in BIoT	Eff. in Dashboard
IS1	50%	71.79%	73.33%
IS2	34.78%	28.21%	20%
IS3	39.13%	35.9%	40%
LLM tool	23.91%	28.21%	26.67%

By analyzing the results specifically related to security requirements, we obtained a different result, since the checklist also focuses on inspecting this particular category of requirements. Table 5 shows the number of security defects identified in each inspected item. In this analysis, we did not use the dashboard subsystem because it does not have nor demand security requirements.

Table 5: Comparison of the number of security defects identified by each inspection session per artifact

Inspection Session	Manager	BIoT
IS1	6	5
IS2	6	3
IS3	5	3
LLM tool	6	4

The results of the analyses focused on security showed that the LLM-based tool had similar effectiveness to human inspectors. Thus, Table 6 presents the results of the effectiveness of applying LLM in the inspection of security requirements. To calculate the effectiveness of the inspection carried out by the LLM-based tool specifically for security, we calculated the number of security defects identified by each study in each artifact, which is divided by the total number of known security defects in each artifact. For example, the inspection session using the LLM-based workflow identified 6 security defects in the Manager artifact; thus, according to the baseline, these 6 identified security defects were divided by 8 (known security defects in the Manager in Table 2), totaling an effectiveness of 75%.

5.2 Discussion

The results indicate that the proposed LLM-based work achieved better defect detection performance in the context of security-related requirements than in the inspection of functional requirements. As we can observe in Table 6, the LLM tool achieved approximately 75% performance in identifying security defects. Some points need to be highlighted for this discussion.

Table 6: LLM tool's security effectiveness in relation to humans

Inspection Session	Eff. in Manager	Eff. in BIoT
IS1	75%	83.3%
IS2	75%	50%
IS3	62.5%	50%
LLM tool	75%	66,67%

- LLM and prompts: The LLM used was chosen for convenience due to its smaller number of parameters and the computational resources we had available. This may have impacted the inspection performance. Furthermore, we need to emphasize that prompts can also influence this performance.
- Checklist: Although the checklist contains questions related to functional requirements, its creation was specifically proposed to identify security defects. Therefore, the number of functional requirements questions is the same as the number of security requirements questions.
- Training: Another point we should highlight is that all human inspectors, despite being novices, received specific training on IoT, requirements engineering, security, and software inspections before performing the inspection sessions.

The obtained results provide evidence to respond to the main research question *"Can a local LLM-based workflow perform functional and security-focused software requirements inspections for IoT systems with performance comparable to human inspectors?"* The answer may be yes, given the current indication we have that the LLM-based workflow, as it has been tailored, supports the identification of functional and security defects in requirements artifacts of IoT written in Brazilian Portuguese. This software technology is intended to be used as initial support in the rapid detection of defects, but it does not replace human inspection; it only intends to assist them in finding defects, not deciding about them.

5.3 Limitations

Despite its promising emerging results, the proposed solution still presents some limitations. The first concerns the computational resources required to execute local models with a large number of parameters. Since the solution was designed for local execution, the available hardware resources directly influence the obtained results. Besides, this study was conducted using IoT software requirements artifacts written in Brazilian Portuguese and worked with only one local LLM model family (Qwen3). Therefore, the obtained results may not generalize to other domains, languages, models, or industrial environments.

Another limitation of this work is the availability of suitable requirements documents for inspection. The existence of a larger and properly structured requirements dataset would enable the investigation of additional model adaptation techniques, such as fine-tuning.

We can also highlight that the comparison baseline was based on inspection sessions conducted with trained undergraduate students rather than domain experts [7], which may have influenced the comparison results. Furthermore, no statistical analysis was performed due to the exploratory nature of this study. Moreover, the comparison baseline, composed of inspections performed by humans, did not provide results that allowed deeper analyses and comparisons, as the study was conducted by novices rather than domain specialists.

6 Final Considerations and Future Work

In this paper, we presented a Local LLM-based workflow to assist or semi-automate the software inspection process, with results comparable to those of human inspectors. The workflow architecture, although tailored to a specific checklist, was built to allow the use of different inspection techniques and approaches by simply configuring its parameters and prompts. The results showed that, despite the generic nature of the workflow, the use of domain-specific prompts proved to be effective in guiding the system toward the execution of a specific task, such as the inspection of security-focused requirements for IoT software systems. To the best of our knowledge, this is one of the first studies investigating local LLMs for security-focused requirements inspection in Brazilian Portuguese.

The application of this work within the traditional software inspection cycle may enable humans and LLMs to collaborate as inspectors, or even allow preliminary inspections to be performed by the artifact authors using the LLM-based workflow to rapidly identify an initial set of defects before entering the formal inspection cycles with human inspectors, because the human presence in the process is still necessary, as the results showed that the LLM is prone to errors.

The current state of the proposed solution sheds light on several future work opportunities. Due to the modular architecture of the workflow, it can be adapted to different software inspection scenarios. Additionally, further studies are required to assess the effectiveness and cost-effectiveness of adopting local LLM-based inspection approaches in different development environments, as well as to investigate their applicability to software system requirements outside the IoT context.

Finally, within the specific context of security inspection for IoT software systems, future work may focus on refining the prompts employed in the workflow, which can lead to evolving the checklist used. Modifications to inspection structures, checklist definitions, and example sets may help reduce ambiguities, minimize false positives and false negatives, and improve the effectiveness of the proposed approach.

Artifact Availability

The workflow proposed in this work is available at: https://doi.org/10.5281/zenodo.20350427

Acknowledgements

The authors thank CNPq and CAPES (Finance Code 001). Prof. Travassos is a CNPq researcher (grant 304234/2018-4) and CNE FAPERJ (grant E-26/204.310/2024).

References

[1] Simon Arvidsson and Johan Axell. 2023. Prompt engineering guidelines for LLMs in Requirements Engineering. Department of Computer Science and Engineering University of Gothenburg.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805. doi:10.1016/j.comnet.2010.05.010

[3] Victor R. Basili and Dieter H. Rombach. 1994. Goal Question Metric (GQM) Approach. *Encyclopedia of software engineering* (1994), 528–532.

[4] Amanda Bertsch, Maor Ivgi, Emily Xiao, Uri Alon, Jonathan Berant, Matthew R. Gormley, and Graham Neubig. 2025. In-Context Learning with Long-Context Models: An In-Depth Exploration. arXiv:2405.00200 [cs.CL] https://arxiv.org/abs/2405.00200

[5] Stefan Biffl and Michael Halling. 2003. Investigating the defect detection effectiveness and cost benefit of nominal inspection teams. *IEEE Transactions on Software Engineering* 29, 5 (2003), 385–397.

[6] Chitrak Biswas and Souvick Das. 2024. AI-agent based requirements inspection and analysis through question answering. *Innovations Syst Softw Eng* (2024).

[7] Jeffrey Carver, Letizia Jaccheri, Sandro Morasca, and Forrest Shull. 2003. Issues in using students in empirical studies in software engineering education. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*. 239–249. doi:10.1109/METRIC.2003.1232471

[8] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2025. Unleashing the potential of prompt engineering for large language models. *Patterns* 6, 6 (2025), 101260. doi:10.1016/j.patter.2025.101260

[9] Bruno Pedraça de Souza. 2025. IoT Security Evaluation: A Set of Strategies Combined of Inspection, Testing, and Experimentation. In *Anais do XXVIII Congresso Ibero-Americano em Engenharia de Software*.

[10] André Giron, Kaway Marinho, Leonardo Drummond, Sabrina Rocha, Larissa Galeno, Rodrigo Gonçalves, and Guilherme Travassos. 2025. Sistema de Software IoT para Monitoramento de Biossegurança de Instalações. 229–234 pages. doi:10.5753/sbcas_estendido.2025.7228

[11] Miguel J. Hornos and Mario Quinde. 2024. Development methodologies for IoT-based systems: challenges and research directions. *Journal of Reliable Intelligent Environments* 10 (2024), 215–244.

[12] Joseph Howarth. [n. d.]. *Number of Parameters in GPT-4 (Latest Data).* Retrieved June 22, 2025 from https://explodingtopics.com/blog/gpt-parameters

[13] Jitendra Jonnagaddala and Zoie Shui-Yee Wong. 2025. Privacy preserving strategies for electronic health records in the era of large language models. npj Digital Medicine.

[14] Marcos Kalinowski and Guilherme H. Travassos. 2004. A computational framework for supporting software inspections. *19th International Conference on Automated Software Engineering* (2004), 46–55.

[15] Qwen LM. [n. d.]. *Qwen3: Think Deeper, Act Faster.* Retrieved August 21, 2025 from https://qwenlm.github.io/blog/qwen3

[16] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? arXiv:2202.12837 [cs.CL] https://arxiv.org/abs/2202.12837

[17] Ollama. [n. d.]. *Models.* Retrieved August 21, 2025 from https://ollama.com/search

[18] Victor Vidigal Ribeiro, Daniela Soares Cruzes, and Guilherme Horta Travassos. 2022. Moderator factors of software security and performance verification. *Journal of Systems and Software* 184 (2022), 111137. doi:10.1016/j.jss.2021.111137

[19] Nicolli Rios, Rodrigo Spínola, and Guilherme Travassos. 2022. Exploring Technical Debt on IoT Software Projects. 88–97 pages. https://sol.sbc.org.br/index.php/sbqs/article/view/23295

[20] Yuya Sasaki, Hironori Washizaki, Jialong Li, Dominik Sander, Nobukazu Yoshioka, and Yoshiaki Fukazawa. 2024. Systematic Literature Review of Prompt Engineering Patterns in Software Engineering. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. 670–675. doi:10.1109/COMPSAC61105.2024.00096

[21] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL] https://arxiv.org/abs/2203.11171

[22] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. 2025. On protecting the data privacy of Large Language Models (LLMs) and LLM agents: A literature review. *High-Confidence Computing* 5, 2 (2025), 100300. doi:10.1016/j.hcc.2025.100300

[23] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2026. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[24] Yongchao Zhou, Andrei I. Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *Eleventh International Conference on Learning Representations* (2022).