Pontifícia Universidade Católica
do Rio de Janeiro

Murilo de Sá Martin

Evaluating the Potential of Large Language Models in Security-Related Software Requirements Classification

Undergraduate Project Work

Undergraduate project work presented to the Undergraduate Program in Computer Engineering of PUC-Rio in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Advisor: Prof. Juliana Alves Pereira

Rio de Janeiro
January 2025

Pontifícia Universidade Católica
do Rio de Janeiro

Murilo de Sá Martin

Evaluating the Potential of Large Language Models in Security-Related Software Requirements Classification

Undergraduate project work presented to the Undergraduate Program in Computer Engineering of PUC-Rio in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering. Approved by the Examination Committee.

Prof. Juliana Alves Pereira
Advisor
Departamento de Informática – PUC-Rio

Prof. Anderson Gonçalves Uchôa
Universidade Federal do Ceará – UFC

Prof.ª Daniel José Barbosa Coutinho
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

Rio de Janeiro, January 7th, 2025

All rights reserved.

Murilo de Sá Martin

Bibliographic data

de Sá Martin, Murilo

Evaluating the Potential of Large Language Models in Security-Related Software Requirements Classification / Murilo de Sá Martin; advisor: Juliana Alves Pereira. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2025.

v., 48 f: il. color. ; 30 cm

Projeto Final de Graduação - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Non-Functional Requirements – Teses. 2. Machine Learning – Teses. 3. Requirements Engineering – Teses. 4. Natural Language Processing – Teses. 5. Requirements Classification – Teses. 6. Engenharia de requisitos;. 7. Requisitos não funcionais;. 8. Modelos de linguagem de grande escala;. 9. Classificação de requisitos de segurança;. 10. Engenharia de prompts;. 11. Aprendizado de máquina;. I. Alves Pereira, Juliana. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I want to express my deep gratitude to my girlfriend, friends and family for their unwavering support and understanding throughout this journey. Writing this thesis often meant sacrificing precious moments with them, and their patience and encouragement were fundamental in helping me persevere.

Also, I am especially thankful to my academic advisor, who not only guided me with expertise but also understood the demanding situation I was in—balancing work, other academic responsibilities and this research. Her understanding and support were invaluable in helping me achieve this milestone.

Abstract

de Sá Martin, Murilo; Alves Pereira, Juliana. **Evaluating the Potential of LLMs in the Classification of Security-Related Software Requirements.** Rio de Janeiro, 2025. 48p. Final Undergraduate Project - Department of Informatics, Pontifical Catholic University of Rio de Janeiro.

Effective classification of security-related software requirements is essential for mitigating potential threats and ensuring robust system design. This study investigates the accuracy of large language models (LLMs) in classifying security-related requirements compared to traditional machine learning (ML) methods. Using the SecReq and PROMISE_exp datasets, we evaluated nine LLMs across various prompt engineering strategies. The re- sults demonstrate that LLMs achieve high accuracy and outperform traditional ML models in several evaluation scenarios and that prompt engineering can significantly enhance the model's ability to identify security-related requirements. This work underscores the domain-generalization capabilities of LLMs and their potential to streamline requirements classification without the complexity of feature engineering or dataset-specific fine-tuning often required by ML approaches. Researchers, practitioners, and tool developers can leverage these findings to advance automated approaches in security requirements engineering.

Keywords

Requirements engineering; Non-functional requirements; Large language models; Security requirements classification; Prompt engineering; Machine learning;

Resumo

de Sá Martin, Murilo; Alves Pereira, Juliana. **Avaliando o Potencial de LLMs na Classificação de Requisitos de Software Relacionados a Segurança** . Rio de Janeiro, 2025. 48p. Projeto Final de Graduação – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A classificação eficaz de requisitos de software relacionados à segurança é essencial para mitigar potenciais ameaças e garantir um design de sistema robusto. Este estudo investiga a precisão dos Modelos de Linguagem de Grande Escala (LLMs) na classificação de requisitos relacionados à segurança em comparação com métodos tradicionais de aprendizado de máquina (ML). Utilizando os conjuntos de dados SecReq e PROMISE_exp, avaliamos nove LLMs em diferentes estratégias de engenharia de prompts. Os resultados demonstram que os LLMs alcançam alta acurácia e superam os modelos tradicionais de ML em diversos cenários de avaliação, além de mostrar que a engenharia de prompts pode melhorar significativamente a capacidade dos modelos de identificar requisitos relacionados à segurança. Este trabalho destaca as capacidades de generalização dos LLMs e seu potencial para simplificar a classificação de requisitos sem a complexidade de engenharia de atributos ou *fine-tuning*, comumente necessários em aborda- gens de ML. Pesquisadores, profissionais e desenvolvedores de ferramentas podem aproveitar essas descobertas para avançar em abordagens automatizadas na engenharia de requisitos de segurança.

Palavras-chave

Engenharia de requisitos; requisitos não funcionais; Modelos de linguagem de grande escala; Classificação de requisitos de segurança; Engenharia de prompts; Aprendizado de máquina;

Table of contents

Table of contents

List of tables

1
Introduction

Software requirements are detailed descriptions of the functions, capabilities, and constraints that a software system must fulfill to meet the needs of its users and stakeholders. They are categorized into functional requirements (FRs) and non-functional requirements (NFRs). FRs detail *what* tasks the system must perform, while NFRs describe *how* a system should operate [1]. These requirements serve as a blueprint for developers, guiding software design, implementation, and testing.

One major challenge in managing NFRs is their inherent diversity and complexity. There is no consensus regarding their definition, scope, level of abstraction, granularity, priority, and inter-dependencies, making it hard for stakeholders to identify, comprehend, and communicate about these requirements [36]. Among existing NFRs, security stands out as particularly critical. In recent years, the field of *Security Requirements Engineering* (SRE) has attracted considerable attention within the requirements engineering community [2]. It is widely accepted that incorporating security early in the development process is more cost-effective and leads to a more robust design [3]. Thus, its early identification and integration would mitigate potential threats and reduce future security-related issues. However, distinguishing security-related requirements from other FRs and NFRs can be tedious and error-prone [38].

This study aims to investigate the accuracy of open and closed-source LLMs in classifying security-related software requirements and compare them with state-of-the-art Machine Learning (ML) approaches documented in the literature. To evaluate the effectiveness of LLMs in classifying security-related NFRs, we used two datasets: SecReq [4, 5] and a subset of the PROMISE_exp [39] dataset. SecReq contains 510 requirements, including 187 security-related and 323 non-security-related requirements. The PROMISE_exp subset, specifically created to categorize security-related and non-security-related requirements, comprises 125 security-related and 844 non-security-related requirements.

This work differs from previous studies by focusing on the use of LLMs for security requirements classification instead of traditional ML algorithms. Traditional ML algorithms often require complex feature engineering, special-

ized skills, and significant setup effort, as well as the creation of large amounts of labeled training data, which is time-consuming and labor-intensive and can make them harder to adopt. In contrast, LLMs offer ease of use through natural language prompting and minimal setup effort. The simplicity and ease of interaction enabled by NLP is a key factor that can make them well-suited for smooth integration with agile software management tools like Jira and others, enhancing their practicality in modern development workflows. LLMs, if proven effective, can be integrated directly within these tools and play a pivotal role in addressing critical security aspects early in the development process. They enable precise task assignments by automatically identifying security-related requirements and allocating them to team members with the appropriate expertise. This not only enhances efficiency but also fosters better collaboration among team members, ensuring that security concerns are addressed with the necessary focus and expertise.

The results demonstrate that, overall, LLMs are effective at accurately identifying security-related requirements, even in a zero-shot setting. Regarding the impact of prompt engineering on the evaluation metrics for each model, most prompting strategies yielded diminishing or negligible improvements. The notable exception was *raw-inst*, a combination of role and instruction prompting (see Section 2.2.1), which significantly enhanced the models' ability in detecting security-related requirements. Contrary to expectations, where more complex prompts were anticipated to have a greater impact on larger models, no clear correlation was observed between model size or version and accuracy improvement with different prompting techniques. In fact, the most substantial improvements were seen using the raw-inst technique (see Section 2.2.1) in the smallest models from the oldest generations of the Mistral and Gemma families, which initially exhibited the weakest results in the baseline zero-shot approach. *Mistral-nemo*, the model with the highest F1_score when evaluated on the SecReq dataset, outperformed the traditional J48 algorithm presented in [6] in almost all scenarios, and outperformed knauss'es Bayesian classifier [7] in cross-domain evaluations. The results highlight the potential of LLMs, particularly when combined with role-and-instruction-based prompting strategies like raw-inst, to achieve solid accuracy in classifying security-related requirements. Our approach not only demonstrates better domain generalization compared to the traditional ML models evaluated on the SecReq dataset but also surpasses them in some intra-domain evaluation scenarios, without the fine-tuning and labeling overhead commonly associated with ML models.

The contributions of this work are as follows:

- A comprehensive study of how 9 different LLMs perform when classifying

security requirements from 1,479 requirement specifications from the SecReq [4, 5] and PROMISE_exp [39] datasets.

- An in-depth analysis of how prompt engineering techniques can be applied in classifying security requirements and how they affect the accuracy and performance of LLMs.
- A comparative evaluation against traditional ML models, highlighting the domain-independent performance of LLMs and their potential as viable alternatives to task-specific models.
- We have made all artifacts from our study publicly available at [8].

Audience: Researchers, practitioners, and tool builders benefit from our experiments and insights in understanding how effective LLMs are in identifying and classifying security requirements in comparison to ML-based approaches. Furthermore, our work showcases the potential of prompt engineering techniques in this field.

2
Background and Related Work

In this section, we define key concepts and explore related studies in the field of software requirements engineering.

2.1
Functional and Non-Functional Requirements

The specification of software requirements is a starting point for software development. Institute of Electrical and Electronics Engineers (IEEE) Standard 610.12-1990 [37] defines software requirements in three ways:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability, as in (1) or (2).

Requirements can be categorized into *Functional Requirements* (FRs) and *Non-Functional Requirements* (NFRs). FRs specify a function that a system or system component must be able to perform [37]. While FRs are generally easier to define, NFRs are less straightforward, as despite the term being used for decades, there is no consensus on its definition [36]. For example, Anton et al. [9] describe NFRs as capturing the non-behavioral aspects of a system, while others emphasize system attributes like reliability, performance, and security [10]; or focus on constraints that guide system operation and development [11]. Although interpretations vary, NFRs generally describe *how* a system should operate, and there is an unanimous consensus that NFRs are important and can be critical for the success of a project [36].

2.2 Large Language Models for Software Engineering

The rapid evolution of *Large Language Models* (LLMs) has been driven by advances in deep learning, abundant computing resources, and vast training datasets, resulting in powerful capabilities for *Natural Language Processing* (NLP) tasks and broad applications across multiple fields [12]. Software Engineering (SE) – a discipline focused on the development, implementation, and maintenance of software systems – is one of those areas reaping the benefits of the LLM revolution [13]. The use of LLMs in SE primarily emerges from an innovative perspective where numerous SE challenges can be effectively reframed into data, code, or text analysis tasks. Within SE, requirements engineering (RE) – a critical subset focused on the identification, analysis, and documentation of software requirements – has seen notable applications of LLMs. A recent systematic literature review by Hou et al. [14] identifies several tasks within the domain of requirements engineering where LLMs have been applied. Among these tasks are anaphoric ambiguity treatment, requirements classification, co-reference detection, and specification generation.

- Anaphoric ambiguity treatment: In requirements engineering, anaphoric ambiguity occurs when a pronoun can plausibly refer to different entities, leading to varying interpretations by different readers [15]. This ambiguity can result in different interpretations of requirements, potentially causing suboptimal software artifacts during development. Ezzini et al. [15] and Sridhara et al. [16] have demonstrated the effectiveness of LLMs like SpanBERT and ChatGPT in addressing anaphoric ambiguity in software requirements.
- Requirements classification: Requirements, which stem from natural language documents, require efficient classification, particularly for early-stage project assessments, such as identifying security-related ones [3]. Hey et al. [17] demonstrate that NoRBERT, a model fine-tuned from BERT, excels in classifying both functional and non-functional requirements, outperforming traditional methods in several tasks. Additionally, Luo et al. [18] propose PRCBERT, a BERT-based classification method that utilizes flexible prompt templates. It achieves accurate requirements' classification for zero-shot scenarios.
- Co-reference detection: Co-reference in Requirements Engineering occurs when different expressions in a requirements document refer to the same system component. If co-references are not resolved, it can lead to ambiguity or misinterpretation of the requirements, causing confusion about

what is being described. Wang et al. [19] propose a fine-tuned version of BERT that accurately detects co-reference in software requirements.

- Specification generation: Specification generation involves creating formal program specifications that define the behavior and functionality of software systems. Manually crafting these specifications is particularly challenging for complex programs, as they must capture all semantic details of the code accurately. Ma et al. [20] introduce SpecGen, a novel technique leveraging LLMs to successfully generate verifiable specifications for most of the programs evaluated in their study. Xie et al. [21] conducted experiments on state-of-the-art LLMs, evaluating their performance and cost-effectiveness for specification generation, and found that certain open-source models achieve high effectiveness in specification generation and not only outperform traditional approaches, but also surpass larger, more expensive closed-source LLMs.

2.2.1 Prompt Engineering

Prompt engineering has become a vital technique for expanding the abilities of LLMs. This technique utilizes task-specific instructions, called prompts, to improve model accuracy without altering the core model parameters. Instead of adjusting these parameters, prompts enable pre-trained models to be smoothly applied to downstream tasks by triggering desired behaviors based solely on the prompt provided [22].

There are various prompting techniques, ranging from simple to complex, and no universally "best" technique exists. The effectiveness of a technique depends on the specific task. Simpler prompting techniques may work better for straightforward tasks, whereas more complex tasks often require additional effort from the user to craft detailed and nuanced prompts. Sahoo et al. [22] conducted a systematic survey of prompt engineering, categorizing various prompting strategies according to their suitability for different task domains. OpenAI provides a page on prompt engineering in its documentation[1], where six strategies for achieving better results with their models are explained. Unlike most academic papers, this documentation targets a broader audience, not necessarily academics or researchers, and thus employs simpler, more accessible language. While it does not provide metrics on how each technique affects model accuracy and is less rigorous than academic papers, the page covers several important prompting techniques referenced in the literature. Common prompting techniques from the literature include:

[1] https://platform.openai.com/docs/guides/prompt-engineering

- *zero_shot*: This technique [23] eliminates the need for extensive training data, relying instead on carefully crafted prompts that guide the model toward novel tasks based solely on the model's pre-existing knowledge.
- *few_shot*: This technique involves giving a model a few examples of a task at the time of use. The model then uses these examples to understand and complete a new, similar task, which reduces the need for large amounts of task-specific data [24].
- *Manual Chain-of-Thought (CoT) Prompting*: This technique provides language models with the ability to generate a coherent series of intermediate reasoning steps that lead to the final answer for a problem. The idea is that LLM can generate chains of thought if demonstrations of chain-of-thought reasoning are provided in the exemplars for few-shot prompting [25].
- *Automatic Chain-of-Thought (auto-CoT) Prompting*: Instead of writing manual reasoning demonstrations one by one for each example as in Manual-CoT, auto-CoT leverages a simple prompt like *"Let's think step by step"* to facilitate step-by-step thinking before answering a question [26].
- *Role Prompting*: This technique involves providing the model with a specific role in the prompt. The assigned role offers context about the LLM's identity and background, enabling it to generate more natural, in-character responses tailored to that role [27]. Clavie et al. [28] explored combining role instructions with detailed task instructions, referring to this approach as *raw-inst*.

2.3 Security Requirements Classification

To develop secure and reliable software systems, security requirements must be analyzed with caution [38]. There are several works in the literature that use ML and Deep Learning (DL) solutions to identify and classify security requirements [7, 6, 38].

Knauss et al. [7] introduced a tool-supported method to identify and categorize security requirements using a Bayesian classifier. They used the SecReq dataset and, according to their experiments, their approach succeeds in assisting requirements engineers in the task of identifying security-relevant requirements. It identifies the majority of the security-relevant requirements, achieving a recall greater than 0.9 and maintaining a precision exceeding 0.8,

indicating only a few false positives. These results were observed when the models were trained and evaluated within the same specification domain.

Security requirements are often mixed with other types of requirements. Thus, many existing methods do not deliver the expected efficiency due to their domain-dependency [38]. To address this limitation, Li [6] proposed a hybrid method for identifying security requirements that combines linguistic analysis with ML. Their approach first revises a conceptual model of security requirements by defining linguistic rules and security keywords from literature. This definition guides the extraction of features for training classifiers using standard ML algorithms. They evaluated the method using a combination of different subsets of the Industry requirements specifications in the SeqReq dataset. Although their approach achieves a good average precision and recall across all tested subsets (0.79 and 0.75, respectively), the benchmark approach – Bayesian classifier trained by Knauss et al. [7] on the same dataset – demonstrates superior accuracy, with an average precision of 0.83 and recall of 0.92. Nonetheless, the classifiers developed by their approach show promising potential for generalization to other domains, achieving an average precision of 0.69 and recall of 0.64 in cross-dataset evaluation, while the benchmark approach achieved an average precision of 0.51 and recall of 0.53.

Armin Kobilica et al. [38] conducted an empirical study evaluating the effectiveness of 22 supervised ML classifiers and 2 deep learning approaches, using the SecReq dataset. Their approach minimized the typical overhead of linguistic and semantic preprocessing by applying simpler text preprocessing techniques. They apply word encoding for most classifiers and word embedding for CNN-based models. The results indicated that the Long Short-Term Memory (LSTM) network achieved the highest accuracy among deep learning models at 84%, while Boosted Ensemble led the supervised classifiers with an accuracy of 80%,demonstrating the strength of simplified preprocessing paired with robust algorithms.

Although existing solutions leveraging ML and DL for security-related requirements classification have shown promising results, they also exhibit certain limitations. One of the main challenges lies in their domain dependency; these models often rely on features or data specific to particular contexts, limiting their applicability across diverse projects. Additionally, such approaches often involve significant overhead, including the need for fine-tuning models, labeling data, and configuring parameters—all of which require considerable expertise and effort.

In contrast, LLMs present a more versatile and user-friendly alternative. Unlike traditional ML or DL approaches, LLMs are not domain-dependent,

enabling broader applicability without extensive customization. Moreover, their ability to process natural language prompts eliminates the need for complex configurations or fine-tuning, making them significantly easier to deploy. These advantages position LLMs as a practical solution for addressing the challenges of security-related requirements' classification, particularly in scenarios where domain generalization and usability are critical.

3
Study Design

This study aims *to investigate the effectiveness and applicability of LLMs for classifying security requirements, focusing specifically on the context of requirement specification documents at the software design stage*. Given the specialized language in the developer description of these documents, our goal is to determine whether LLMs can reach a higher accuracy than traditional ML-based approaches and to understand how prompt engineering could improve their performance.

3.1
Research Questions

Based on our aim, this study is guided by three research questions (RQs), as follows:

RQ1 *What is the accuracy of LLMs in distinguishing security-related requirements from non-security requirements in a zero-shot approach?*

RQ2 *How do different prompting strategies change the accuracy of LLMs in classifying security-related requirements?*

RQ3 *How does the accuracy of LLMs compare to the accuracy of models reported in studies from the literature?*

RQ1 evaluates the baseline accuracy of LLMs when no additional task-specific data is provided. The goal is to assess their capability to classify security-related requirements based solely on general pre-trained knowledge without any further fine-tuning or task-specific training. We will evaluate the models using a zero-shot prompting strategy (see Section 3.2.4). The model's predictions will be compared to the ground truth labels, and metrics such as precision, recall, and F1 score will be calculated to assess accuracy. RQ2 investigates whether and how varying the way instructions (prompts) are presented to the LLM affects its classification accuracy. We used 3 additional prompt engineering strategies (see Section 3.2.4). Each strategy was evaluated on the same dataset, and the accuracy metrics were compared to identify the most effective approach. RQ3 aims to benchmark the accuracy of LLMs against existing ML-based models specifically developed for classifying security-related

requirements, as reported in prior research. We identified relevant studies that have addressed similar classification tasks. Then, we compared key metrics (precision, recal and F1-score) of these studies. The LLM's accuracy, evaluated in RQ1 and RQ2, was analyzed to determine whether LLMs can serve as viable alternatives or if task-specific models still outperform them in this domain.

3.2 Study Steps

Our study consists of seven steps, which are illustrated in Figure 3.1. They are also described in the following sections.

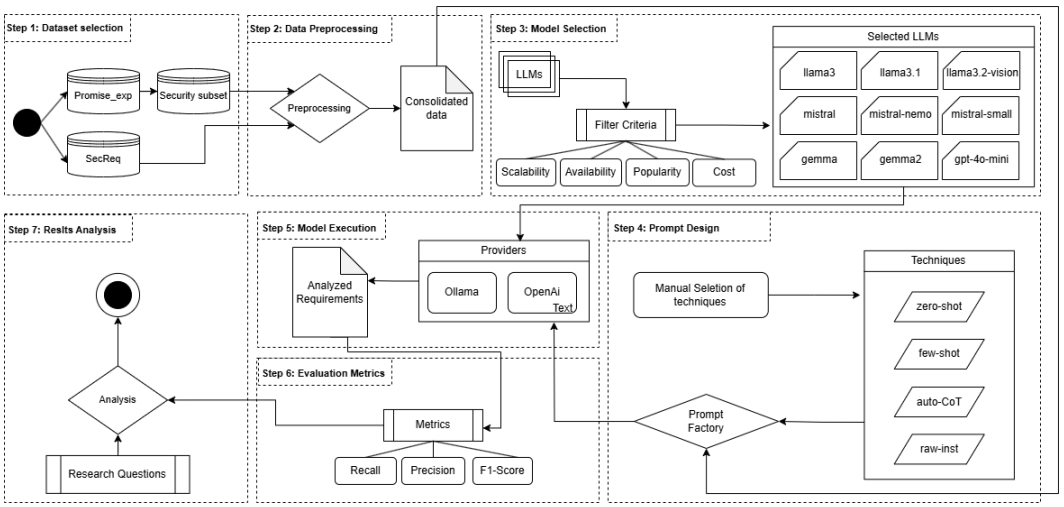

Figure 3.1: Overview of the study methodology.

3.2.1 Dataset Selection

One major challenge in the field of software requirements engineering is the lack of publicly available, high-quality datasets [29]. This scarcity of reliable data limits the development and evaluation of models designed for tasks such as requirements classification. In this paper, we leverage two datasets, PROMISE_exp [39] and SecReq [4, 5], to support the classification of security-related requirements.

The PROMISE_exp dataset, developed by Márcia Lima [39], is an expanded version of the publicly available PROMISE dataset, originally hosted in the PROMISE dataset repository and introduced in studies such as [34] and [35]. While the original PROMISE dataset contains 625 labeled software requirements from 15 different projects, PROMISE_exp increases the number

of requirements to 969 by incorporating 34 new projects into the dataset. These requirements are categorized into several types, including Availability (A), Fault Tolerance (FT), Legal (L), Look & Feel (LF), Maintainability (MN), Operational (O), Performance (PE), Portability (PO), Scalability (SC), Security (SE), Usability (US), and Functional (F). The expanded dataset includes 125 security-related requirements, with the remaining 844 categorized as non-security-related to form the security-focused subset used in this study. The expansion process involved structured web searches based on the keyword *"Software Requirement Specification"*, manual analysis of the search results, and extraction of textual requirements in English. Structural compatibility with the original dataset was carefully maintained, ensuring continuity for researchers who previously worked with the original PROMISE dataset.

SecReq is a dataset developed by Schneider et al. [4, 5], comprising 510 requirements, of which 187 are security-related. The dataset includes three industry standards: Common Electronic Purse (ePurse) with 124 labeled entries, Customer Premises Network (CPN) with 210 labeled entries, and Global Platform Specification (GP) with 176 labeled entries. The requirements are manually labeled by experts as either "sec" (security-related) or "non-sec" (non-security-related).

3.2.2 Data Preprocessing

Given that more than a thousand requirements need to be evaluated, with each requirement assessed multiple times, establishing an automated testing pipeline is essential. The PROMISE_exp dataset is in .arff format, while the SecReq dataset is in .csv format. A Python script was developed to extract each requirement along with its corresponding label, consolidating them into a unified JSON file. The JSON file contains an array of objects, where each object represents a requirement with the following fields:

- *project_id*: Identifier for the project. The PROMISE_exp dataset already had project identifiers, ranging from 1 to 49, and these were maintained. The three industry standards (CPN, ePurse, and GPS) in the SecReq dataset received the IDs 50, 51, and 52, respectively.
- *requirement*: The text of the requirement.
- *label*: The classification of the requirement, either "sec" (security-related) or "nonsec" (non-security-related).
- *source*: The origin of the requirement, such as "PROMISE_exp" or "secreq".

During data processing, one entry was found to be invalid in the SecReq dataset. While most requirements are labeled as "sec" or "nonsec", one had an erroneous label, "XYZ". This entry was removed from the consolidated JSON file. After processing the data, the consolidated dataset had 1,167 nonsec instances (78.9%) and 312 sec instances (21.1%).

3.2.3 Model Selection

The field of LLMs is advancing rapidly, with new, cutting-edge models emerging on a monthly basis. This continuous innovation makes it a highly dynamic period for artificial intelligence and LLM research. Given the large number of available models, careful selection was necessary for this study.

Our pool of models consists of 4 families: Gemma, Mistral, Llama and GPT. In terms of popularity, the first three chosen families of models – Gemma, Mistral, and Llama – are part of the top 20 most popular (in terms of downloads) families of LLM available in hugging face[1]. Due to cost concerns, we initially did not plan to test any closed-source models. However, during the development of this study, the release of the *GPT-4o mini* model—promising exceptional affordability and speed—prompted its inclusion in our pool of tested models. The *GPT-4o mini* model represented a true paradigm shift in cloud LLM usage, with reports indicating that the number of active users in OpenAI APIs doubled after its release [30].

The inclusion of both open-source and closed-source models allowed us to explore their contrasting characteristics. While closed-source models are typically executed through cloud-based APIs, which offload the computational overhead to the provider's infrastructure, open-source models can be deployed locally, granting users full control over their execution environment. Closed-source models benefit from managed services, including uptime guarantees, automatic updates, and scalability, enabling developers to focus solely on utilizing the API. However, this reliance comes with drawbacks, such as potential privacy concerns, data dependence on the provider, and the ongoing operational costs associated with API usage. In contrast, open-source models, when deployed locally, require users to handle the hardware setup, maintenance, and updates but offer enhanced data privacy, customization, and cost predictability. In our case, the closed-source cloud-deployed model utilized was *GPT-4o mini*, which provided all the advantages mentioned previously with minimal drawbacks. For our use case, there were no privacy concerns, and the cost of

[1]https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads

the cloud-based service was extremely low. On the other hand, the locally deployed LLMs were free to use but required significant computational resources and substantially more time to run all the prompts.

Table 3.1 describes the names and characteristics of the nine models selected in this study. In terms of scalability, all models chosen are available in different variations, which mainly affect parameter count and performance, especially inference speed and memory requirements. A higher parameter count can also be associated with higher reasoning capabilities. For each model, we selected the configuration with the highest parameter count that our setup could support. The models tested in this study ranged from 7 billion to 27 billion parameters. Approximately 10 billion parameters is generally considered the cutoff for a “small” language model [31]. While OpenAI does not disclose the exact parameter count of GPT-4o mini, it is described as a "small model" on their website [32]. However, without additional details, it remains unclear whether GPT-4o mini falls below this 10 billion parameter threshold.

Table 3.1: Characteristics of the chosen LLMs

Name	**Parameters (in billions)**	**Size (GB)**	**License Type**
Llama 3	8	4.7	Open Source
Llama 3.1	8	4.9	Open Source
Llama 3.2 - Vision	11	7.9	Open Source
Mistral	7.25	4.1	Open Source
Mistral-Nemo	12	7.1	Open Source
Mistral-Small	22	12.0	Non-commercial use only
Gemma	7	5.0	Open Source
Gemma2	27	15.0	Open Source
GPT-4o mini	-	-	Closed Source

3.2.4 Prompt Design

In terms of prompt design, we began with the zero-shot prompt, which served as the baseline for RQ_1. For RQ_2 and RQ_3, we additionaly selected 3 prompting strategies: few-shot, auto-CoT, and raw-inst (see Section 2.2.1). These strategies were chosen based on their simplicity, making them straightforward to implement in practical scenarios, and their frequent use in prior studies [23, 24, 26, 27, 28] .

The evaluation involved testing multiple prompting strategies across thousands of software requirement specifications, making it essential to automate prompt generation. To address this, we developed a script that functioned

as a prompt factory, dynamically injecting requirements from our consolidated file (Step 2 in Figure 3.1) into predefined templates.

Every prompt template instructs the model to classify a given requirement as either security-related or non-security-related. Another common feature across all prompts is the inclusion of an answer template, essential for automating evaluation and ensuring output consistency by specifying the response format in JSON. The rest of the prompt's structure varies depending on the strategy used. The zero-shot prompt (Appendix .1.2, Item 1) has the simplest structure, including only the requirement, task specification, and answer template, with no additional information provided. For the few-shot prompt[2] (Appendix .1.2, Item 2), where we provide a set of examples for the model, markers like *"user_x"* and *"response_x"* distinguish each example, guiding the model to follow a consistent pattern. In the auto-CoT prompts (Appendix .1.2, Item 3), additional instructions prompt the model to think through its response in structured steps before delivering the final answer, encouraging a reasoning-driven approach. For the raw-inst prompt (Appendix .1.2, Item 4), the model is provided with a detailed description of its role and the task to be performed.

3.2.5 Model Execution

We decided to use two providers to run the selected LLMs: OpenAI[3] (cloud-based) and Ollama[4] (local). We use Ollama tool to deploy *Llama 3*, *llama3.1*, and *llama3.2-vision* (from Meta), *Mistral*, *Mistral-Nemo*, and *Mistral-Small* (from Mistral AI), *Gemma*, and *Gemma2* (from Google). The OpenAI API was used to deploy the *GPT-4o mini*.

The locally-deployed LLMs were executed in a computer with the following specifications: AMD Ryzen 5 5600X 6-Core Processor, 16GB of RAM and an NVIDIA GeForce RTX 3060 GPU, with 12GB of VRAM. We tune the model to be as deterministic as possible experimenting with different parameters and using *temperature=0*, thus removing the need for multiple runs using each prompt.

[2]In our implementation of the few-shot strategy, we provided one example from each class (security-related and non-security-related). Note that it could be considered a "one-shot" approach by some authors.

[3]https://openai.com/index/openai-api

[4]https://ollama.com

3.2.6 Evaluation Metrics

We will utilize the confusion matrix and its derived metrics to compare the various prompts and models evaluated in this study. The confusion matrix is a table summarizing the model's predictions compared to the actual labeled instances as "sec" or "nonsec". The confusion matrix can be represented as follows:

	Predicted "sec"	Predicted "nonsec"
Actual "sec"	True Positive (TP)	False Negative (FN)
Actual "nonsec"	False Positive (FP)	True Negative (TN)

Where:

- **True Positive (TP)**: Instances where the predicted requirement is classified as security-related, and it is also labeled for experts as security-related.
- **False Positive (FP)**: Instances where the model incorrectly predicts a requirement as security-related, but it is labeled as non-security-related.
- **False Negative (FN)**: Instances where the model fails to predict a requirement as security-related, even though it is actually labeled as security-related.
- **True Negative (TN)**: Instances where the model correctly predicts a requirement as non-security-related, and it is also labeled by experts as non-security-related.

Several important metrics are derived from the confusion matrix:

Precision Precision is the proportion of correctly predicted positive instances out of all instances predicted as positive:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall Recall, also known as sensitivity, is the proportion of actual positive instances that were correctly identified by the model:

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-Score The F1-score is the harmonic mean of precision and recall, balancing the trade-off between these metrics:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

These metrics, derived from the confusion matrix, enable a detailed evaluation of the models' accuracy and facilitates the comparison of different LLMs, prompting strategies and ML models in Section 4.

We utilized Python scripts (available as part of the replication package [8]) that leveraged the pandas library[5] to analyze the output of the LLMs in terms of these metrics.

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

4
Results and Discussion

In this section, we discuss the results of the research questions defined in Section 3.

4.1
Accuracy of Zero Shot Approach (RQ_1)

Table 4.1 presents the precision, recall and F1-score for the nine models evaluated on their ability to classify security-related requirements using the baseline prompting approach, *zero-shot*. Overall, all models exhibited high recall but struggled with precision, reflecting their strong ability to identify true positives while often misclassifying false positives. F1-scores varied widely, ranging from 0.38 to 0.77, highlighting the models' varying abilities to balance these metrics.

Among the models, *mistral-nemo* stood out with the highest F1-score of 0.77, driven by its strong precision (0.69) and recall (0.88), showcasing superior capability in identifying security-related requirements. *Mistral-small* and *mistral* achieve comparable results, both with F1-scores of 0.70 and high recall (0.97–0.98). *Llama3*, *Llama3.1* and *llama3.2-vision* followed closely, with F1-scores ranging from 0.72 to 0.73, demonstrating balanced ability in identifying security-related requirements. These models maintain strong precision (0.58–0.60) and recall (0.93–0.96), indicating they are reliable classifiers with balanced results. At the other end of the spectrum, *gemma* recorded the lowest F1-score of 0.38, driven by its extremely low precision (0.23), revealing a tendency to over-classify instances as *sec* and introduce more false positives.

The cloud-based model, *GPT-4o-mini*, achieved a F1-score of 0.70. Its F1-score is competitive with other well-performing models like *Llama3* and *Mistral* variants, making it a strong candidate for scenarios where maximizing recall is critical. However, it does not match the balance achieved by *Mistral-nemo*, which delivers both higher precision and F1-score. As *GPT-4o-mini* tends to sacrifice precision (0.55), it may not be the ideal choice when false positives carry a significant cost. Thus, leading to require additional manual verification to confirm the classifications.

Finding 1. *Mistral-nemo* emerged as the most effective model in the *zero-shot* approach, with the Llama3 series delivering consistently strong results.

model	precision	recall	f1-score
gemma	0.23	**1.00**	0.38
gemma2	0.51	0.97	0.67
gpt-4o-mini	0.55	0.99	0.70
llama3	0.58	0.96	0.72
llama3.1	0.60	0.93	0.73
llama3.2-vision	0.60	0.95	0.73
mistral	0.52	0.97	0.68
mistral-nemo	**0.69**	0.88	**0.77**
mistral-small	0.55	0.98	0.70

Table 4.1: Acuracy by model in zero-shot approach

4.2 Accuracy of Different Prompting Engineering Strategies (RQ_2)

Table 4.2 summarizes the accuracy (precision, recall and F1-score) for the nine models and three additional prompting strategies investigated in this research question.

Few-shot. This technique generally underperformed compared to the baseline (*zero-shot*) approach, with an average F1-score difference of -0.09 (see Table 4.3). Only one model, *gemma*, which had the poorest accuracy in the zero-shot approach, showed improvement, but its F1-score (0.43) remained below competitive levels. For all other models, precision and recall worsened, with some experiencing an F1-score drop of up to 0.20, underscoring the limitations of this strategy for this task.

Auto-CoT. The auto-CoT technique showed limited impact on the models' overall accuracy. Changes in F1-score were minimal, ranging from a negligible increase (+0.01 for *mistral-nemo*) to a slight decrease (-0.06 for *gemma2*), resulting in an average difference of -0.02 (see Table 4.3). *Mistral-nemo*, the most effective model in RQ_1, exhibited a marginal improvement, achieving an

F1-score of 0.78. However, the improvements were insufficient to make auto-CoT a compelling strategy for enhancing classification effectiveness.

Raw-inst. This technique had the most substantial impact on the effectiveness of the models. The F1-score differences ranged from +0.19 to +0.03, with an average of +0.09 (see Table 4.3). Apart from *gemma*, all models achieved accuracy comparable to or exceeding the most-accurate model from RQ_1. *Mistral* emerged as the top performer under this strategy, achieving an F1-score of 0.82, driven by high precision (0.77) and recall (0.88). *GPT-4o-mini* followed closely, with an F1-score of 0.81, supported by a precision of 0.70 and recall of 0.96. *Mistral-small* also demonstrated a strong balance between precision (0.82) and recall (0.79), resulting in an F1-score of 0.80. These results highlight raw-inst as a promising approach for improving model accuracy.

Finding 2. Most prompting strategies did not enhance the models' ability to classify security-related requirements. However, the raw-instruction prompting strategy proved to be the most effective, with every model showing an increase in F1-score and an average F1-score improvement of +0.09.

4.3 Comparison with State-of-the-art Approaches (RQ_3)

Mistral-nemo with the raw-inst technique, which proved to be the most effective on the SecReq dataset, was defined as our baseline (see Table 4.4). We compare the accuracy of the baseline model to two studies [7, 6] that utilized state-of-the-art ML models for the same task.

strategy	model	precision	recall	f1-score
few-shot	gemma	0.27	**1.00**	0.43
	gemma2	0.43	**1.00**	0.60
	gpt-4o-mini	0.49	0.99	0.65
	llama3	0.39	**1.00**	0.56
	llama3.1	0.36	**1.00**	0.53
	llama3.2-vision	0.40	0.99	0.57
	mistral	0.44	**1.00**	0.61
	mistral-nemo	**0.55**	0.95	**0.69**
	mistral-small	0.50	0.99	0.67
auto-CoT	gemma	0.23	**1.00**	0.38
	gemma2	0.45	0.98	0.61
	gpt-4o-mini	0.54	0.98	0.70
	llama3	0.56	0.96	0.71
	llama3.1	0.52	0.96	0.68
	llama3.2-vision	0.54	0.96	0.69
	mistral	0.50	0.98	0.67
	mistral-nemo	**0.71**	0.87	**0.78**
	mistral-small	0.54	0.99	0.70
raw-inst	gemma	0.40	**0.99**	0.57
	gemma2	0.67	0.94	0.78
	gpt-4o-mini	0.70	0.96	0.81
	llama3	0.66	0.95	0.78
	llama3.1	0.64	0.94	0.76
	llama3.2-vision	0.63	0.95	0.76
	mistral	0.77	0.88	**0.82**
	mistral-nemo	**0.82**	0.79	0.80
	mistral-small	0.68	0.94	0.79

Table 4.2: Accuracy by prompting strategy and model

strategy	model	precision diff	recall diff	f1 diff
few-shot	gemma	**0.04**	0.00	**0.05**
	gemma2	-0.08	0.03	-0.07
	gpt-4o-mini	-0.06	0.00	-0.05
	llama3	-0.19	0.04	-0.16
	llama3.1	-0.24	**0.07**	-0.20
	llama3.2-vision	-0.20	0.04	-0.16
	mistral	-0.08	0.03	-0.07
	mistral-nemo	-0.14	**0.07**	-0.08
	mistral-small	-0.05	0.01	-0.03
	Average	-0.11	0.03	-0.09
auto-CoT	gemma	0.00	0.00	0.00
	gemma2	-0.06	0.01	-0.06
	gpt-4o-mini	-0.01	-0.01	0.00
	llama3	-0.02	0.00	-0.01
	llama3.1	-0.08	**0.03**	-0.05
	llama3.2-vision	-0.06	0.01	-0.04
	mistral	-0.02	0.01	-0.01
	mistral-nemo	**0.02**	-0.01	**0.01**
	mistral-small	-0.01	0.01	0.00
	Average	-0.03	0.01	-0.02
raw-inst	gemma	0.17	-0.01	**0.19**
	gemma2	0.16	-0.03	0.11
	gpt-4o-mini	0.15	-0.03	0.11
	llama3	0.08	-0.01	0.06
	llama3.1	0.04	**0.01**	0.03
	llama3.2-vision	0.03	0.00	0.03
	mistral	**0.25**	-0.09	0.14
	mistral-nemo	0.13	-0.09	0.03
	mistral-small	0.13	-0.04	0.09
	Average	0.13	-0.03	0.09

Table 4.3: Difference in accuracy by prompting strategy compared to the baseline approach

model	strategy	precision	recall	f1-score
mistral-nemo	raw-inst	**0.79**	0.84	**0.82**
gpt-4o-mini	raw-inst	0.69	**0.96**	0.80
mistral	raw-inst	0.72	0.88	0.80
gemma2	raw-inst	0.66	0.94	0.77
llama3	raw-inst	0.63	0.94	0.76

Table 4.4: 5 most accurate models and strategies for the SecReq dataset

Table 4.5 presents a comparative analysis of the precision, recall, and F1-score metrics from Studies 1 [7] and 2 [6] alongside the results of our study. For Studies 1 and 2, the table includes performance metrics under two distinct evaluation scenarios:

- *Intra-domain evaluation*, where the models were trained and tested within the same specification domain, providing insights into their accuracy in a controlled and consistent context.
- *Cross-domain evaluation*, where the models were tested on data from different domains, showcasing their generalizability. Both the best-case and worst-case accuracy metrics are reported to capture the variability and robustness of the models across diverse datasets.

This comparative approach highlights the strengths and limitations of each study, emphasizing how domain-specific or cross-domain factors influence the effectiveness of the classification models.

Study 1, conducted by Knauss et al. [7], reported strong results, with F1-scores of 0.88 for the ePurse specification, 0.86 for the GP specification, and 0.96 for the CPN specification for the intra-domain evaluation (see Table 4.5). However, F1-score dropped significantly for cross-domain evaluations. For the ePurse specification, the F1-score dropped to 0.58 in the best-case scenario and further decreased to 0.47 in the worst-case scenario. For the GP specification, the F1-score dropped to 0.57 and 0.23 in the best and worst-case scenarios, respectively. For the CPN specification, the F1-score fell sharply to 0.40 in the best-case scenario and 0.33 in the worst-case scenario

Study 2, conducted by Li [6], achieved an F1-score of 0.82 for the ePurse specification, 0.74 for the GP specification and 0.73 for the CPN specification for the intra-domain evaluation (see Table 4.5). Similar to Study 1, F1-score generally dropped when evaluating models trained on other specification domains, *i.e.* cross-domain evaluations. However, for the ePurse specification, the F1-score increased to 0.88 in the best-case scenario and decreased to 0.65 in the worst-case scenario. For the GP specification, the F1-score dropped to 0.61 and 0.29 in the best and worst-case scenarios, respectively. For the CPN specification, the F1-score dropped to 0.63 in the best-case scenario and 0.6 in the worst-case scenario.

Specification	Study	Precision	Recall	F1-score
ePurse	Ours	0.98	0.77	0.86
	Study 1 (intra-domain)	0.83	**0.93**	**0.88**
	Study 1 (cross-domain best case)	0.72	0.48	0.58
	Study 1 (cross-domain worst case)	**0.99**	0.33	0.47
	Study 2 (intra-domain)	0.90	0.75	0.82
	Study 2 (cross-domain best case)	0.94	0.82	**0.88**
	Study 2 (cross-domain worst case)	0.95	0.49	0.65
GP	Ours	0.71	**0.95**	0.82
	Study 1 (intra-domain)	0.81	0.92	**0.86**
	Study 1 (cross-domain best case)	0.43	0.85	0.57
	Study 1 (cross-domain worst case)	0.29	0.19	0.23
	Study 2 (intra-domain)	0.79	0.70	0.74
	Study 2 (cross-domain best case)	0.50	0.78	0.61
	Study 2 (cross-domain worst case)	**0.85**	0.17	0.29
CPN	Ours	0.68	0.83	0.75
	Study 1 (intra-domain)	**0.98**	**0.95**	**0.96**
	Study 1 (cross-domain best case)	0.29	0.65	0.40
	Study 1 (cross-domain worst case)	0.23	0.54	0.33
	Study 2 (intra-domain)	0.76	0.71	0.73
	Study 2 (cross-domain best case)	0.52	0.78	0.63
	Study 2 (cross-domain worst case)	0.50	0.76	0.60

Table 4.5: Comparison of our results with Studies 1 and 2 for intra-domain and cross-domain evaluation.

Study 1 achieved high scores when models were trained and tested on the same dataset, outperforming our approach in all specifications. The differences in F1-scores were small for the ePurse and GP specifications (0.02 and 0.04, respectively), but considerable for the CPN specification (0.21). Their results dropped significantly in cross-domain evaluations, whereas our approach achieved higher F1-scores across all specifications by a large margin. The difference in F1-scores in the best-case scenario was 0.28 for the ePurse specification, 0.25 for the GP specification, and 0.35 for the CPN specification. Our approach also outperformed Study 2 in most cases. In the intra-domain evaluation scenario, the F1-score of our model surpassed theirs by 0.04 for the ePurse specification, 0.08 for the GP specification, and 0.02 for the CPN specification. In the cross-domain evaluation scenario, considering the best-case scenario, our model had a lower F1-score for the ePurse specification (0.86 compared to 0.88), but higher F1-scores for the GP specification (0.82 compared to 0.61) and for the CPN specification (0.75 compared to 0.63).

Overall, one major limitation of state-of-the-art techniques is that ML-based models often require retraining for each new dataset to adapt to its specific characteristics and requirements. This retraining process requires a substantial amount of manual effort to label the requirements in the new dataset, introducing additional time and cost burdens. Moreover, the need for domain expertise during the labeling process can further increase the complexity and expense. The reliance on fine-tuning also makes cross-domain approaches less flexible and scalable, as each dataset essentially demands a custom model configuration.

In contrast, LLMs excel in this aspect, as they can operate effectively without fine-tuning. Leveraging their pre-trained knowledge, LLMs have demonstrated impressive results across various specifications, often surpassing traditional approaches in both accuracy for cross-domain evaluations and ease of deployment. This eliminates the need for dataset-specific training, significantly reducing the manual overhead and cost associated with traditional ML techniques. Furthermore, ML techniques are prone to overfitting to the training data, which can lead to a steep decline in accuracy when models encounter datasets with varying structures or terminology. These limitations highlight the practical challenges of state-of-the-art techniques, reinforcing the advantages of LLMs in providing robust, cost-effective, and scalable solutions.

Finding 3. LLMs can serve as viable alternatives to task-specific state-of-the-art ML models. Our approach outperforms both state-of-the-art models in terms of F1-score across most specifications and evaluation scenarios. It demonstrates more robust and less domain-dependent performance compared to prior studies, yielding results comparable to those of domain-specific models.

4.4 Threats to Validity

The validity of this study is influenced by several factors that could impact the reliability and generalizability of the findings. This section highlights potential threats to validity and reflects on the measures taken to mitigate them, as well as the limitations that remain.

Internal validity. Internal validity relates to whether the results accurately reflect the performance of the evaluated models, independent of external influences. The study relied only on two datasets, PROMISE_exp and SecReq. We knowledge that limitations in the data quality and labeling could have introduced biases that affected the evaluation results. To mitigate such threat to the validity of our results, these datasets were selected due to their extensive use in prior research and their established reputation for robustness and reliability. Their use also supports the reproducibility of our findings, as they provide a common ground for comparison with other works in the literature. Moreover, requirements from both datasets were provided sequentially within the same session, potentially allowing the models to carry context or learn patterns from one dataset that might influence its performance on the other. Although responses were later isolated to evaluate performance on specific datasets, the sequential prompting approach may have inadvertently affected the model's behavior during evaluation. Additionally, the study tested a limited number of prompting strategies.While these strategies were adequate for initial evaluation, the rapid development of LLMs has introduced an ever-growing pool of possible prompts and techniques that were not explored in this study. This limitation suggests that a broader variety of prompting strategies could yield different or more comprehensive insights into the models' capabilities. Expanding the range of prompts considered in future research could mitigate this limitation and enhance the robustness of the evaluation.

External validity. External validity pertains to the generalizability of the study's findings to real-world contexts, beyond the specific datasets and experimental conditions used in this research. This study focused on two datasets, PROMISE_exp and SecReq, which represent a particular subset of software requirements, with a special emphasis on security-related ones. While these datasets are useful for evaluating the classification capabilities of LLMs in the context of security requirements, they may not fully capture the diversity of software requirements in broader domains or industries. Therefore, the results may not necessarily generalize to other types of NFRs from different domains.

Construct validity. Construct validity focuses on whether the evaluation methods and metrics effectively measure the intended objectives. This study employed standard classification metrics such as precision, recall, and F1-score, which are well-established and widely accepted in similar research domains. However, the reliance on manual prompt engineering introduces a degree of subjectivity that could impact the consistency of results. Slight variations in prompt phrasing can lead to differences in performance. To address this limitation and ensure transparency, we have made all the constructed prompts publicly available. These prompts were carefully designed and refined based on initial model responses to maximize clarity and relevance while aligning with the study's objectives. The availability of these prompts allows for systematic evaluation of how different phrasing or structures impact the performance of the models, contributing to a deeper understanding of the role of prompt engineering in such studies. Furthermore, the study did not explore fine-tuning the model on domain-specific data. This decision aligns with the growing interest in leveraging the out-of-the-box capabilities of LLMs. By avoiding fine-tuning, we make our methodology broadly applicable and resource-efficient, as fine-tuning can be computationally expensive and may limit generalizability. Additionally, our findings demonstrate that high-quality results can be achieved without fine-tuning.

5
Conclusion and Future work

This study evaluates the efficacy of LLMs in classifying security-related software requirements, comparing their performance to that of state-of-the-art ML algorithms. Our study highlights several key implications and contributions for developers, tool builders, and researchers, which are as follows:

- **LLMs are effective for security-related requirement classification:** The zero-shot approach demonstrated that LLMs, particularly models like *mistral-nemo*, achieved high accuracy in identifying security-related requirements, with an F1-score of 0.77. This indicates that LLMs can perform the classification task effectively without the need for domain-specific training or significant overhead.
- **Prompt engineering can significantly improve model accuracy:** The application of prompting strategies, particularly a hybrid approach combining Role Prompting and Instruction Prompting, referred to as *raw-inst*, led to a marked improvement in model accuracy. The raw-inst approach increased the F1-score by an average of +0.09, with models like *Mistral* achieving an F1-score of 0.82. This demonstrates the impact that refined prompting can have on enhancing classification outcomes, with minimal overhead.
- **LLMs can outperform traditional ML models:** When compared to state-of-the-art ML models, *Mistral-nemo* outperformed the traditional J48 algorithm presented in [6] in almost all scenarios, and surpassed Knauss's Bayesian classifier [7] in cross-dataset evaluations.

The findings of this study open up numerous promising avenues for future research:

- **Use of the PROMISE+ dataset:** Future work should incorporate the PROMISE+ dataset, developed by Silva et al. [33], which is an expansion of the PROMISE_exp dataset used in this study. This would enable a more robust evaluation of LLMs in comparison to traditional ML models, providing a broader and more representative sample of security-related software requirements.

- **Comparison with ML models using both datasets:** In this study, we compared LLMs to ML models using only the SecReq dataset. To strengthen this comparison, future work could involve reaching out to the authors of studies mentioned research question 3, and applying the same ML models to the PROMISE_exp. This would provide a more comprehensive evaluation.
- **Comparison with other studies evaluating LLMs for the same task:** This study focused on comparing LLMs only with traditional machine learning models from the literature. Future work could extend this comparison to include other studies that evaluate LLMs for requirements classification tasks, particularly fine-tuned versions of the BERT model, such as NoRBERT [17] and PRCBERT [18], which have demonstrated promising results in related scenarios. This would provide a broader perspective on the effectiveness of LLMs in this domain.
- **Expansion of models and prompting strategy pool:** The set of LLMs and prompting strategies used in this study was limited. Future research should explore a wider range of models and prompting strategies, including variations in phrasing, context, and detail. This would help identify the most effective approaches for security-related software requirements classification and take advantage of recent advancements in the field.
- **Real-time requirement classification:** Building on the findings of our study, we plan to develop an integrated tool designed to classify requirements specifications in real-time. This tool would seamlessly integrate with existing requirement specification platforms, such as JIRA, IBM DOORS, or other popular tools used by software development teams. The proposed tool could implement the best strategies identified in our findings, leveraging multiple LLMs based on the available computational resources. It would provide actionable insights directly within the environment where requirements are managed, making it easier to address critical security aspects during the early stages of development. Furthermore, incorporating automated security requirement classification can assist assigning tasks to qualified team members.
- **Qualitative analysis of the datasets:** Future work should incorporate a qualitative analysis of the requirements in both datasets. This analysis would allow researchers to identify the strengths and weaknesses of the LLMs' classification accuracy. Additionally, it would help pinpoint areas where prompting strategies could be improved, ultimately enhancing

the overall accuracy. A more thorough qualitative evaluation of the requirements themselves could also provide insights into potential biases or gaps in the data, offering further avenues for refinement of the models and strategies used.

6
Data Availability

All scripts and data used in this study are available in the replication package [8]. The replication package provide comprehensive instructions for setting up and reproducing the experiments. This includes: (1) the specific configurations, hyperparameters, and memory requirements for each LLM; (2) information on how our prompt was built; and (3) scripts for reproducing the evaluations, including comments to guide replication.

7
Appendix

.1
Prompting templates

In this appendix, we present the placeholders used in the prompt strategy along with their corresponding values, as well as the templates utilized for each strategy.

.1.1
Placeholders

- **Target requirement**: Requirement to be classified
- **Security requirement**: Example requirement with the label sec
- **Non-security requirement**: Example requirement with the label non-sec
- **Task specification**: "Label it as a security-related requirement (sec) or non-security-related requirement (nonsec)."
- **Answer template specification**: "Return the result as a JSON with the following format: {{label: sec or nonsec}}"
- **CoT promotion**: "Let's think step by step. Provide reasoning before giving the response."
- **Model instruction**: "You are an expert in requirements engineering. You are tasked with the classification of requirements for a software project. You should consider 2 types of requirements: security-related requirement (sec) and non security-related requirements (nonsec)."
- **Task instruction**: "Security-related requirements are those that explicitly address the protection of a system's data, resources, and functionalities from unauthorized access, threats, or vulnerabilities. They encompass aspects such as user authentication, data encryption, access controls, and compliance with security standards. In contrast, non-security-related requirements pertain to the general functionality and performance of a system without specific considerations for security. These may include operational features, usability, and system performance metrics that do not inherently involve safeguarding against security risks."

.1.2 Templates

The exact placeholders values are described in Section .1.1.

1. Zero-Shot:
   "For the given requirement: {*target requirement*}, {*task specification*}. {*answer template specification*}"

2. Few-Shot:
   "user_1 = For the given requirement: {*security requirement*}, {*task specification*}. {*answer template specification*}
   response_1: {'label': 'sec'}
   user_2 = For the given requirement: {*non security requirement*}, {*task specification*}. {*answer template specification*}
   response_2: {'label': 'nonsec'}
   user_3 = For the given requirement: {*target requirement*}, {*task specification*}. {*answer template specification*}.
   response_3: ..."

3. Auto-CoT:
   "For the given requirement: {*target requirement*}, {*task specification*}. {*CoT promotion*}. {*answer template specification*}."

4. Raw-Inst:
   "{*model instruction*}. {*task instruction*}. For the given requirement: {*target requirement*}, {*task specification*}. {*answer template specification*}.

Bibliography

[1] SOMMERVILLE, I.. **Engenharia de Software**. Pearson, 2013.

[2] MELLADO, D.; BLANCO, C.; SÁNCHEZ, L. E. ; FERNÁNDEZ-MEDINA, E.. **A systematic review of security requirements engineering**. Computer Standards & Interfaces, 32(4):153–165, 2010.

[3] KIM, H.-K.; CHUNG, Y.-K.. **Automatic translation form requirements model into use cases modeling on uml**. In: COMPUTATIONAL SCIENCE AND ITS APPLICATIONS–ICCSA 2005: INTERNATIONAL CONFERENCE, SINGAPORE, MAY 9-12, 2005, PROCEEDINGS, PART III 5, p. 769–777. Springer, 2005.

[4] HOUMB, S. H.; ISLAM, S.; KNAUSS, E.; JÜRJENS, J. ; SCHNEIDER, K.. **Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec**. Requirements Engineering, 15:63–93, 2010.

[5] SCHNEIDER, K.; KNAUSS, E.; HOUMB, S.; ISLAM, S. ; JÜRJENS, J.. **Enhancing security requirements engineering by organizational learning**. Requirements Engineering, 17:35–56, 2012.

[6] LI, T.. **Identifying security requirements based on linguistic analysis and machine learning**. In: 2017 24TH ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC), p. 388–397. IEEE, 2017.

[7] KNAUSS, E.; HOUMB, S.; SCHNEIDER, K.; ISLAM, S. ; JÜRJENS, J.. **Supporting requirements engineers in recognising security issues**. In: REQUIREMENTS ENGINEERING: FOUNDATION FOR SOFTWARE QUALITY: 17TH INTERNATIONAL WORKING CONFERENCE, REFSQ 2011, ESSEN, GERMANY, MARCH 28-30, 2011. PROCEEDINGS 17, p. 4–18. Springer, 2011.

[8] MARTIN, M.. **Replication Package for LLM for Security-Related Requirement Classification**. https://github.com/aisepucrio/llm-security-req-classification.git, dec 2024. Accessed: 2024-12-30.

[9] ANTON, A. I.. **Goal identification and refinement in the specification of software-based information systems**. Georgia Institute of Technology, 1997.

[10] DAVIS, A. M.. **Software requirements: objects, functions, and states**. Prentice-Hall, Inc., 1993.

[11] KOTONYA, G.; SOMMERVILLE, I.. **Requirements engineering: processes and techniques**. Wiley Publishing, 1998.

[12] RAIAAN, M. A. K.; MUKTA, M. S. H.; FATEMA, K.; FAHAD, N. M.; SAKIB, S.; MIM, M. M. J.; AHMAD, J.; ALI, M. E. ; AZAM, S.. **A review on large language models: Architectures, applications, taxonomies, open issues and challenges**. IEEE Access, 2024.

[13] MA, W.; LIU, S.; LIN, Z.; WANG, W.; HU, Q.; LIU, Y.; ZHANG, C.; NIE, L.; LI, L. ; LIU, Y.. **Lms: Understanding code syntax and semantics for code analysis**. arXiv preprint arXiv:2305.12138, 2023.

[14] HOU, X.; ZHAO, Y.; LIU, Y.; YANG, Z.; WANG, K.; LI, L.; LUO, X.; LO, D.; GRUNDY, J. ; WANG, H.. **Large language models for software engineering: A systematic literature review**. ACM Transactions on Software Engineering and Methodology, 2023.

[15] EZZINI, S.; ABUALHAIJA, S.; ARORA, C. ; SABETZADEH, M.. **Automated handling of anaphoric ambiguity in requirements: a multi-solution study**. In: PROCEEDINGS OF THE 44TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 187–199, 2022.

[16] SRIDHARA, G.; MAZUMDAR, S. ; OTHERS. **Chatgpt: A study on its utility for ubiquitous software engineering tasks**. arXiv preprint arXiv:2305.16837, 2023.

[17] HEY, T.; KEIM, J.; KOZIOLEK, A. ; TICHY, W. F.. **Norbert: Transfer learning for requirements classification**. In: 2020 IEEE 28TH INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE (RE), p. 169–179. IEEE, 2020.

[18] LUO, X.; XUE, Y.; XING, Z. ; SUN, J.. **Prcbert: Prompt learning for requirement classification using bert-based pretrained language models**. In: PROCEEDINGS OF THE 37TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, p. 1–13, 2022.

[19] WANG, Y.; SHI, L.; LI, M.; WANG, Q. ; YANG, Y.. **A deep context-wise method for coreference detection in natural language requirements**. In: 2020 IEEE 28TH INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE (RE), p. 180–191. IEEE, 2020.

[20] MA, L.; LIU, S.; LI, Y.; XIE, X. ; BU, L.. **Specgen: Automated generation of formal program specifications via large language models**. arXiv preprint arXiv:2401.08807, 2024.

[21] XIE, D.; YOO, B.; JIANG, N.; KIM, M.; TAN, L.; ZHANG, X. ; LEE, J. S.. **Impact of large language models on generating software specifications**. arXiv preprint arXiv:2306.03324, 2023.

[22] SAHOO, P.; SINGH, A. K.; SAHA, S.; JAIN, V.; MONDAL, S. ; CHADHA, A.. **A systematic survey of prompt engineering in large language models: Techniques and applications**. arXiv preprint arXiv:2402.07927, 2024.

[23] RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; SUTSKEVER, I. ; OTHERS. **Language models are unsupervised multitask learners**. OpenAI blog, 1(8):9, 2019.

[24] BROWN, T.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J. D.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A. ; OTHERS. **Language models are few-shot learners**. Advances in neural information processing systems, 33:1877–1901, 2020.

[25] WEI, J.; WANG, X.; SCHUURMANS, D.; BOSMA, M.; XIA, F.; CHI, E.; LE, Q. V.; ZHOU, D. ; OTHERS. **Chain-of-thought prompting elicits reasoning in large language models**. Advances in neural information processing systems, 35:24824–24837, 2022.

[26] ZHANG, Z.; ZHANG, A.; LI, M. ; SMOLA, A.. **Automatic chain of thought prompting in large language models**. arXiv preprint arXiv:2210.03493, 2022.

[27] KONG, A.; ZHAO, S.; CHEN, H.; LI, Q.; QIN, Y.; SUN, R.; ZHOU, X.; WANG, E. ; DONG, X.. **Better zero-shot reasoning with role-play prompting**. arXiv preprint arXiv:2308.07702, 2023.

[28] CLAVIÉ, B.; CICEU, A.; NAYLOR, F.; SOULIÉ, G. ; BRIGHTWELL, T.. **Large language models in the workplace: A case study on prompt**

engineering for job type classification. In: INTERNATIONAL CONFERENCE ON APPLICATIONS OF NATURAL LANGUAGE TO INFORMATION SYSTEMS, p. 3–17. Springer, 2023.

[29] LIMAYLLA-LUNAREJO, M.-I.; CONDORI-FERNANDEZ, N. ; LUACES, M. R.. **Towards a fair dataset for non-functional requirements**. In: PROCEEDINGS OF THE 38TH ACM/SIGAPP SYMPOSIUM ON APPLIED COMPUTING, p. 1414–1421, 2023.

[30] REUTERS. **Openai says chatgpt's weekly users have grown to 200 million**. https://www.reuters.com/technology/artificial-intelligence/openai-says-chatgpts-weekly-users-have-grown-200-million-2024-08-29/. Accessed: 2024-12-27.

[31] ZHOU, Z.; LI, L.; CHEN, X. ; LI, A.. **Mini-giants:" small" language models and open source win-win**. arXiv preprint arXiv:2307.08189, 2023.

[32] OPENAI. **Gpt-4o mini: advancing cost-efficient intelligence**. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024. Accessed: 2024-12-15.

[33] SILVA, B.; NASCIMENTO, R.; RIVERO, L.; BRAZ, G.; SANTOS, R.; MARTINS, L. ; VIANA, D.. **Promise+: expandindo a base de dados de requisitos de software promise exp**. Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software, p. 291–301, 2024.

[34] CLELAND-HUANG, J.; SETTIMI, R.; ZOU, X. ; SOLC, P.. **The detection and classification of non-functional requirements with application to early aspects**. In: 14TH IEEE INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE (RE'06), p. 39–48, 2006.

[35] CLELAND-HUANG, J.; SETTIMI, R.; ZOU, X. ; SOLC, P.. **Automated classification of non-functional requirements**. Requirements Engineering, 12(2):103–120, 2007.

[36] GLINZ, M.. **On non-functional requirements**. In: 15TH IEEE INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE (RE 2007), p. 21–26. IEEE, 2007.

[37] **Ieee standard glossary of software engineering terminology**. IEEE Std 610.12-1990, p. 1–84, 1990.

[38] KOBILICA, A.; AYUB, M. ; HASSINE, J.. **Automated identification of security requirements: A machine learning approach**. In: PROCEEDINGS OF THE 24TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, EASE '20, p. 475–480, New York, NY, USA, 2020. Association for Computing Machinery.

[39] LIMA, M.; VALLE, V.; COSTA, E. A.; LIRA, F. ; GADELHA, B.. **Software engineering repositories: Expanding the promise database**. In: PROCEEDINGS OF THE XXXIII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, SBES '19, p. 427–436, New York, NY, USA, 2019. Association for Computing Machinery.