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

Murilo Martin
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
mmartin@aise.inf.puc-rio.br

Daniel Coutinho
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
dcoutinho@inf.puc-rio.br

Anderson Uchôa
Federal University of Ceará (UFC)
Itapajé, Brazil
andersonuchoa@ufc.br

Juliana Alves Pereira
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
juliana@inf.puc-rio.br

ABSTRACT

An effective classification of security-related software requirements is crucial to mitigate potential threats and ensure robust system design. This study investigates the performance of Large Language Models (LLMs) in classifying security-related requirements compared to traditional Machine Learning (ML) methods. Using the SecReq, DOSSPRE and PROMISE+ datasets, we evaluated ten LLMs across various prompt engineering strategies. The results demonstrate that LLMs achieve high accuracy and outperform traditional ML-based 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-based approaches. Researchers, practitioners, and tool developers can leverage these findings to advance automated approaches in security requirements engineering.

KEYWORDS

Requirements Engineering, Large Language Models, Security, Non-Functional Requirements

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 [34]. 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 interdependencies, making it hard for stakeholders to identify, comprehend, and communicate about them [8]. 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* (RE) community [26]. It is widely accepted that incorporating security early in the development process is more cost-effective and leads to a more robust design [14]. 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 [16].

This study aims to investigate the performance 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 requirements, we used three datasets: SecReq [11, 32], DOSSPRE [13], and a subset of the PROMISE+ [33] dataset. The SecReq dataset, specifically created to categorize security-related requirements, contains a total of 510 requirements, including 187 security-related and 323 non-security-related requirements. The DOSSPRE dataset, in particular its binary version, comprises 1,316 requirements, 514 labeled as security-related and 802 labeled as non-security related. Finally, the PROMISE+ subset comprises a total of 3,677 requirements, including 237 security-related and 3,440 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, specialized 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 (Natural Language Processing) 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.

Our results demonstrate that LLMs are effective in identifying security-related requirements. 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' performance. 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 performance improvement with different prompting techniques. The most substantial improvements were seen using the *raw-inst* strategy in the *DeepSeek* model. *DeepSeek* was also the model with the highest F1-Score when evaluated on the SecReq

dataset, outperforming the traditional J48 algorithm presented by Li [19] in almost all scenarios, and outperformed knauss'es Bayesian classifier [15] 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 performance in classifying security-related requirements. Our approach not only demonstrates better domain generalization compared to the traditional ML-based 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: *(i)* a comprehensive study of how ten different LLMs perform when classifying security-related requirements from 5,503 requirement specifications from the SecReq [11, 32], DOSSPRE [13], and PROMISE+ [33] datasets; *(ii)* an in-depth analysis of how prompt engineering strategies can be applied in classifying security requirements and how they affect the performance of LLMs; and, *(iii)* a comparative evaluation against traditional ML models, highlighting the domain-independent performance of LLMs and their potential as viable alternatives to task-specific models.

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

The specification of software requirements is a starting point for software development. Institute of Electrical and Electronics Engineers (IEEE) Standard 610.12-1990 [12] 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; and (3) a documented representation of a condition or capability, as in (1) or (2).

The software 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 [12]. 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 [8]. For instance, Anton et al. [1] describe NFRs as capturing the non-behavioral aspects of a system, while others emphasize system attributes like reliability, performance, and security [6]; or focus on constraints that guide system operation and development [18]. Although interpretations vary, NFRs generally describe *how* a system should operate, and there is a unanimous consensus that NFRs are important and can be critical for the success of a project [8].

2.2 LLMs 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 [29]. 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 [24]. 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 has seen notable applications of LLMs. A recent systematic literature review by Hou et al. [10] 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, which we describe as follows:

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 [7]. This ambiguity can result in different interpretations of requirements, potentially causing suboptimal software artifacts during development. Ezzini et al. [7] and Sridhara et al. [35] 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 [14]. Hey et al. [9] demonstrate that NoRBERT, a model fine-tuned from BERT, excels in classifying both FRs and NFRs, outperforming traditional methods in several tasks. Additionally, Luo et al. [22] 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. [36] 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. [23] introduce SpecGen, a novel technique leveraging LLMs to successfully generate verifiable specifications. Xie et al. [38] conducted experiments on state-of-the-art LLMs, evaluating their performance and cost-effectiveness for specification generation. They 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.

These studies demonstrate the growing promise of LLMs in supporting automation, improving quality, and reducing ambiguity

in requirements engineering. Our study builds upon this foundation by proposing a systematic, large-scale benchmark of general-purpose LLMs across multiple architectural families and prompting strategies for the specific task of classifying security-related requirements. We chose to focus on security requirements to build on prior ML-based work (see Section 2.3). This enabled direct comparison with existing baselines, ensured replicability, and allowed for fair benchmarking using robust datasets. Moreover, compared to other NFRs, security requirements tend to be more implicit and context-sensitive, making them a challenging use case for testing LLM classification capabilities. Lastly, numerous studies have shown that failure to identify and address security requirements can have severe real-world consequences, reinforcing the relevance of this research.

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 performance 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 [31].

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. [31] 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 performance, the page covers several important prompting techniques referenced in the literature. Common prompting techniques from the literature include:

- **Zero_shot** [28] which 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** [2] that involves giving a model a few examples of a task. The model uses these examples to understand and complete a new, similar task, which reduces the need for large amounts of task-specific data.
- **Manual Chain-of-Thought (CoT)** [37] which 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.
- **Automatic Chain-of-Thought (Auto-CoT)** instead of writing manual reasoning demonstrations one by one for each example as in Manual-CoT, auto-CoT [39] leverages a simple prompt like *"Let's think step by step"* to facilitate step-by-step thinking before answering a question.
- **Role Prompting** 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 [17]. Clavie et al. [3] explored combining role instructions with detailed task instructions, referring to this approach as *Raw-inst*.

We systematically compare different prompt strategies in Section 4.2. Our findings show that prompt engineering plays a significant role in maximizing the effectiveness of LLMs and should be considered a core part of any deployment strategy involving these models in SE settings.

2.3 Security Requirements Classification

To develop secure and reliable software systems, security requirements must be analyzed with caution [16]. There are several works in the literature that use ML and Deep Learning (DL) solutions to identify and classify security requirements [15, 16, 19].

Knauss et al. [15] 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 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.

Li [19] 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 the 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 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. [15] on the same dataset – demonstrates superior performance, with an average precision of 0.83 and recall of 0.92. Nevertheless, in cross-dataset evaluation, the classifiers developed in [19] achieved an average precision of 0.69 and recall of 0.64, while the benchmark approach in [15] achieved an average precision of 0.51 and recall of 0.53. Thus, both approaches did not show promising potential for generalization to other domains.

Armin Kobilica et al. [16] conducted an empirical study evaluating the effectiveness of 22 supervised ML classifiers and 2 DL 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 DL models at 84%, while Boosted Ensemble led the supervised classifiers

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

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. 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 [16]. 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.

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 Settings

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 performance than traditional ML-based approaches and to understand how prompt engineering could improve their performance. Based on our aim, this study is guided by three research questions (RQs) as follows:

RQ_1. What is the performance of LLMs in distinguishing security-related requirements from non-security requirements in a zero-shot approach? This question evaluates the baseline performance 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 4.1). 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 performance.

RQ_2. How do different prompting strategies change the performance of LLMs in classifying security-related requirements? This question investigates whether and how varying the way instructions (prompts) are presented to the LLM affects its classification performance. We used three additional prompt engineering strategies beyond zero-shot prompting (see Section 4.2). Each strategy was evaluated on the same dataset, and the performance metrics were compared to identify the most effective approach.

RQ_3. How does the performance of LLMs compare to the performance of state-of-the-art ML-based models? This question aims to benchmark the performance of LLMs against existing ML-based models specifically developed for classifying security-related requirements, as reported in previous research (see Section 2.3). First, we identified relevant studies that have addressed similar classification tasks: [15, 19]. Note that we excluded Armin Kobilica et al. [16] from our comparative analysis mainly for three reasons: (1) Our study adopts cross-project evaluation to assess generalization, whereas [16] uses random splits, potentially inflating results. (2) The authors did not release code or data, preventing fair replication under our experimental setup. (3) As a short paper, [16] lacks depth and uses only accuracy, a poor fit for imbalanced datasets like SecReq. Second, we compared key metrics (precision, recall and F1-score) of these studies. The LLM's performance, evaluated in RQ_1 and RQ_2, was analyzed to determine whether LLMs can serve as viable alternatives or if task-specific models still outperform them in this domain.

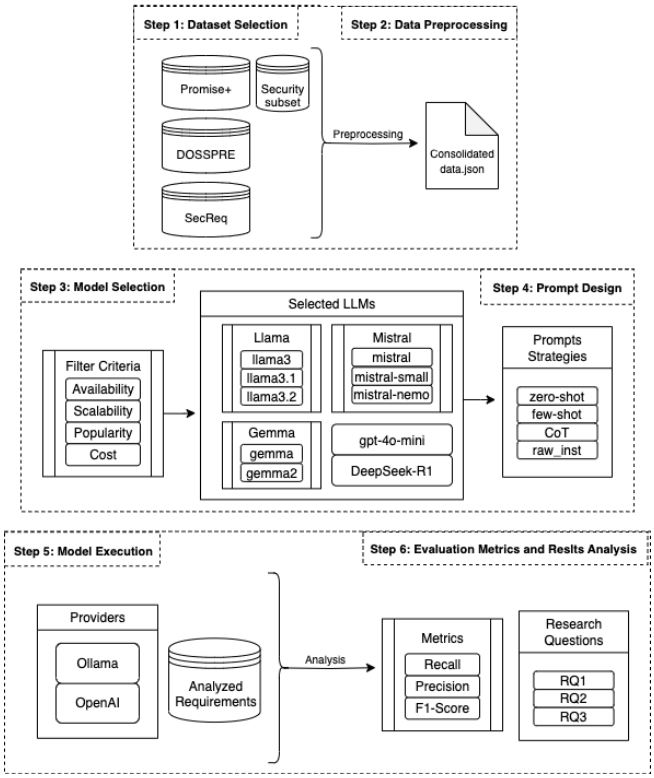

Figure 1: Overview of the Study Methodology

3.1 Study Steps and Procedures

Figure 1 illustrates the six steps of our study. We describe each step and its related procedures as follows.

Step 1: Dataset Selection. One major challenge in the field of software requirements engineering is the lack of publicly available, high-quality datasets [21]. This scarcity of reliable data limits the development and evaluation of models designed for tasks such as requirements classification. In this paper, we leverage three datasets, PROMISE+ [33], DOSSPRE [13] and SecReq [11, 32], to support the classification of security-related requirements. Our choice to use these datasets was driven by their established reputation, extensive prior use, diversity of requirement sources, and the specific objective of RQ_3. Thus, providing a consistent baseline for comparison.

The PROMISE+ dataset, developed by Silva et al. [33], is an expanded version of the PROMISE_exp [20] dataset, which in turn is an expanded version of the PROMISE [4, 5] dataset. While the original PROMISE dataset contains 625 labeled requirements from 15 different projects, PROMISE_exp increases the number of requirements to 969 by incorporating 34 new projects into the dataset. PROMISE+, in turn, increases this number to 3,677 requirements. 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). PROMISE+ dataset includes 237 security-related requirements, with the remaining 3,440 categorized as non-security-related requirements 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, which was then evaluated by five experts.

SecReq is a dataset developed by Schneider et al. [11, 32], 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).

DOSSPRE is a dataset developed by Kadebu et al. [13]. It is available in two versions, one suitable for multi-class classification, and the other (which was the one utilized in this paper), suitable for binary classification. It contains 1,316 security requirements, which were mined from student projects by utilizing a novel process that considers certain maintainability requirements as security-related. From the full set of requirements, 514 are security related and 802 are non-security related.

Step 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+ dataset is in .arff format, while the SecReq and DOSSPRE datasets are 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+ and DOSSPRE datasets already had project identifiers. The three industry standards (CPN, ePurse, and GPS) in the SecReq dataset each had a project_id assigned to it. We utilized the original project_id to create new consolidated IDs, which are sequential between the three datasets.
- *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+", "SecRec", or "DOSSPRE".

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 4,565 "nonsec" instances (83%) and 938 "sec" instances (17%).

Step 3: Model Selection. The field of LLMs is advancing rapidly, with new, cutting-edge models emerging monthly. 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. The selection was designed to ensure a diversity of architectural families and sizes, practical feasibility, and allow for a comparison between open and closed ecosystems. We prioritized popular open-source models from Hugging Face due to their maturity and replicability, and initially focused on them to manage budget constraints. Later, with the release of GPT-4o mini, an affordable, high-performance closed-source model, we included it to enrich our comparison. Finally, model sizes were chosen based on our hardware limitations, enabling us to compare small (smaller than 10B) and large models (up to 27B) and assess the impact of scale on classification performance.

Our pool of models consists of five families: DeepSeek, Llama, Mistral, Gemma, and GPT. In terms of popularity, the first four chosen families of models – DeepSeek, Llama, Mistral, and Gemma – are part of the top 20 most popular (in terms of downloads) families of LLM available in hugging face[2]. Due to cost concerns, we initially did not plan to test any closed-source models. However, 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.

Table 1 describes the names and characteristics of the ten models selected in this study. In terms of scalability, most of the 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 to 27 billion parameters. Approximately 10 billion parameters is generally considered the cutoff for a "small" language model [40]. While OpenAI does not disclose the exact parameter count of GPT-4o mini, it is described as a "small model" on their website [27]. However, without additional details, it remains unclear whether GPT-4o mini falls below this 10 billion parameter threshold.

Step 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 additionally selected three prompting strategies: Few-shot, Auto-CoT, and Raw-inst (see Section 2.2.1). These strategies were chosen based on their simplicity, which makes them straightforward to implement in practical scenarios, and their frequent use in previous studies [2, 3, 17, 28, 39].

[2]https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads. Accessed on April 4, 2025.

Table 1: Characteristics of the Chosen LLMs

Name	Parameters (in billions)	Size (GB)	License Type
Gemma	7	5.0	Open Source
Gemma2	27	15.0	Open Source
Llama 3	8	4.7	Open Source
Llama 3.1	8	4.9	Open Source
Llama 3.2	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
DeepSeek-R1-Distill-Qwen	14	9.0	Open Source
GPT-4o mini	-	-	Closed Source

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 has the simplest structure, including only the requirement description, task specification, and answer template, with no additional information provided. For the few-shot prompt, 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. For the auto-CoT prompt, additional instructions stimulated 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, the model is provided with a detailed description of its role and the task to be performed.

To automate the evaluation process, we developed a script that functioned as a prompt factory, dynamically injecting requirements from our consolidated data file (Step 2 in Figure 1) into the predefined templates. The full prompts and all scripts are available in the replication package [25].

Step 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 *DeepSeek* (from DeepSeek AI), *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 *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.

Step 6: Evaluation Metrics and Results Analysis. We will utilize the confusion matrix and its derived metrics to compare the various prompt strategies 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 by 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 (see Table 2):

- *Precision*: It is the proportion of correctly predicted positive instances out of all instances predicted as positive.
- *Recall*: It is the proportion of actual positive instances that were correctly identified by the model.
- *F1-Score*: It is the harmonic mean of precision and recall, balancing the trade-off between these metrics.

Table 2: Evaluation Metrics Used in this Study

Metric	Precision	Recall	F1-Score
Formula	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

These metrics enable a detailed evaluation of the models' performance and facilitate the comparison of different LLMs, prompting strategies, and ML-based models in Section 4. We used Python scripts (available as part of the replication package [25]) that leveraged the Pandas library[5] to analyze our results.

4 Results and Discussion

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

4.1 Performance of the Zero-Shot Strategy (RQ_1)

Table 3 presents the precision, recall, and F1-score for the ten models evaluated on the task of classifying security-related requirements using three prompting strategies. In this section, we focus on the results obtained using the baseline *zero-shot* strategy. This baseline serves as a valuable reference point for comparing model performance in the absence of tailored prompt engineering.

A clear pattern emerges from the results: *while most models achieve high recall, they generally struggle with precision.* This trade-off suggests that the models are effective at capturing true positives

[3]https://openai.com/index/openai-api
[4]https://ollama.com
[5]https://pandas.pydata.org/

Table 3: Performance of LLMs Across Prompting Strategies

Strategy	Model	Precision	Recall	F1-Score
Zero-shot	gemma	0.18	**0.98**	0.30
	gemma2	0.34	0.86	0.49
	llama3	0.46	0.84	0.59
	llama3.1	0.48	0.81	0.60
	llama3.2	0.27	0.90	0.42
	mistral	0.36	0.85	0.51
	mistral-nemo	**0.54**	0.73	**0.62**
	mistral-small	0.45	0.80	0.57
	deepseek-r1	0.47	0.84	0.60
	gpt-4o-mini	0.42	0.85	0.56
Few-shot	gemma	0.20	**0.94**	0.33
	gemma2	0.33	0.86	0.48
	llama3	0.31	0.89	0.46
	llama3.1	0.28	0.91	0.43
	llama3.2	0.32	0.89	0.47
	mistral	0.33	0.88	0.49
	mistral-nemo	0.42	0.80	0.55
	mistral-small	**0.44**	0.86	**0.58**
	deepseek-r1	**0.44**	0.83	0.57
	gpt-4o-mini	0.39	0.86	0.54
Auto-CoT	gemma	0.18	**0.99**	0.30
	gemma2	0.31	0.87	0.45
	llama3	0.45	0.83	0.58
	llama3.1	0.44	0.83	0.57
	llama3.2	0.27	0.91	0.42
	mistral	0.36	0.85	0.51
	mistral-nemo	**0.56**	0.70	**0.62**
	mistral-small	0.49	0.77	0.60
	deepseek-r1	0.34	0.85	0.48
	gpt-4o-mini	0.42	0.86	0.56
Raw-inst	gemma	0.27	**0.88**	0.41
	gemma2	0.55	0.79	0.65
	llama3	0.54	0.80	0.64
	llama3.1	0.53	0.78	0.63
	llama3.2	0.51	0.80	0.62
	mistral	0.61	0.69	0.65
	mistral-nemo	**0.70**	0.62	0.65
	mistral-small	0.53	0.79	0.64
	deepseek-r1	0.68	0.66	**0.67**
	gpt-4o-mini	0.56	0.79	0.66

(*i.e.*, actual security requirements), but they also produce a substantial number of false positives, misclassifying non-security requirements as security-related. Consequently, the F1-scores – which reflect the balance between precision and recall – vary considerably, ranging from 0.30 to 0.62, depending on the model's ability to control this trade-off.

The best-performing model in this setting is *mistral-nemo*, which achieved an F1-score of 0.62, the highest among all models. Its superior performance is primarily driven by its relatively high precision (0.54), indicating a better ability to avoid false positives while still maintaining a reasonable recall (0.73). This balance makes *mistral-nemo* a strong candidate for applications where over-classification could lead to unnecessary overhead or risk, such as in security triage workflows or compliance audits.

Other models that performed competitively include *llama3*, *llama3.1*, and *deepseek-r1*, all with F1-scores of 0.59 or 0.60, combining recall of 0.81 or 0.84 with moderate precision. At the opposite end, *gemma* recorded the lowest F1-score (0.30), due to its very low precision (0.18). While it achieved the highest recall (0.98), this came at the cost of a significant number of false positives. Such behavior may be tolerable in exploratory phases or safety-critical environments, but less tolerable in production pipelines that demand higher classification precision.

The cloud-based *gpt-4o-mini* achieved an F1-score of 0.56, with recall (0.85) comparable to the top-performing models, but lower precision (0.42). While its overall performance is acceptable, the lower precision means that its use in scenarios with a high cost of false positives would require downstream filtering or manual review. In contrast, models like *mistral-nemo*, which can be deployed locally and still achieve stronger precision, offer more practical advantages in such contexts.

> **Finding 1:** In the zero-shot setting, most models achieved consistently high recall but varied widely in precision. *Mistral-nemo* achieved a stronger balance between both metrics and delivered the best overall F1-score, indicating superior performance.

4.2 Performance of Different Prompting Engineering Strategies (RQ_2)

Table 3 also presents the classification performance for the ten models across three additional prompting strategies: *Few-shot*, *Auto-CoT*, and *Raw-inst* (see Section 2.2.1). To better understand the relative effectiveness of each strategy, we compare their performance against the baseline zero-shot results from RQ_1.

Few-shot. The few-shot prompting strategy showed inconsistent performance across the models. On average, it led to a -0.04 reduction in F1-score, indicating that this strategy generally underperformed compared to the baseline zero-shot strategy. While a few models such as *gemma*, *llama3.2*, and *mistral-small* showed slight gains in F1-score, most models experienced notable drops. The *llama3.1*, for instance, lost 0.17 in F1-score due to a significant drop in precision (-0.20), even though it gained recall. These outcomes suggest that few-shot examples may introduce noise or distract from the core classification task, especially when the examples are not well aligned with the target domain.

Auto-CoT. The Auto-CoT strategy (automatic chain-of-thought prompting) yielded only marginal changes in performance, with an average F1-score difference of -0.02. For most models, including *gemma*, *llama3.2*, *mistral*, and *gpt-4o-mini*, F1-scores remained unchanged. The *mistral-nemo* model — the top performer in RQ_1 — maintained its F1-score of 0.62 under Auto-CoT, showing no added benefit from the strategy. In some cases, such as *deepseek-r1*, performance dropped noticeably (–0.12 in F1-score), driven by a loss in precision. Overall, Auto-CoT did not substantially enhance classification effectiveness and may introduce unnecessary complexity for this task.

Raw-inst. The raw instruction prompting strategy demonstrated the most substantial and consistent improvement, with an average +0.10 increase in F1-score across models. It significantly enhanced model precision (average increase of +0.15), even though recall slightly decreased on average (–0.09). The top performer under

Table 4: Difference in Performance by Prompting Strategy Compared to the Baseline Zero-Shot Prompting

Strategy	Model	Precision	Recall	F1-Score
Few-shot	gemma	0.02	-0.04	0.03
	gemma2	-0.01	0.00	-0.01
	llama3	-0.15	0.05	-0.13
	llama3.1	-0.20	0.10	-0.17
	llama3.2	0.05	-0.01	0.05
	mistral	-0.03	0.03	-0.02
	mistral-nemo	-0.12	0.07	-0.07
	mistral-small	-0.01	0.06	0.01
	deepseek-r1	-0.03	-0.01	-0.03
	gpt-4o-mini	-0.03	0.01	-0.02
	Average	-0.05	0.03	-0.04
Auto-CoT	gemma	0.00	0.01	0.00
	gemma2	-0.03	0.01	-0.04
	llama3	-0.01	-0.01	-0.01
	llama3.1	-0.04	0.02	-0.03
	llama3.2	0.00	0.01	0.00
	mistral	0.00	0.00	0.00
	mistral-nemo	0.02	-0.03	0.00
	mistral-small	0.04	-0.03	0.03
	deepseek-r1	-0.13	0.01	-0.12
	gpt-4o-mini	0.00	0.01	0.00
	Average	-0.02	-0.00	-0.02
Raw_inst	gemma	0.09	-0.10	0.11
	gemma2	0.21	-0.07	0.16
	llama3	0.08	-0.04	0.05
	llama3.1	0.05	-0.03	0.03
	llama3.2	0.24	-0.10	0.20
	mistral	0.25	-0.16	0.14
	mistral-nemo	0.16	-0.11	0.03
	mistral-small	0.08	-0.01	0.07
	deepseek-r1	0.21	-0.18	0.07
	gpt-4o-mini	0.14	-0.06	0.10
	Average	0.15	-0.09	0.10

Table 5: Mann–Whitney U Test Results Comparing F1-Score Distributions Across Prompting Strategies

Comparison	U Statistic	p-value	Significant
Raw-inst vs. Few-shot	9.0	0.0022	Yes
Raw-inst vs. Auto-CoT	9.5	0.0024	Yes
Few-shot vs. Auto-CoT	40.5	0.496	No

raw-inst was *deepseek-r1*, with an F1-score of 0.67, supported by strong precision (0.68) and balanced recall (0.66). This result suggest consistent reliability in identifying relevant requirements without excessive over-classification. Importantly, all models (except *gemma*) achieved F1-scores above 0.62, surpassing the best baseline result from RQ_1.

To further validate this observation and assess whether the differences observed across prompting strategies are statistically significant, we conducted statistical analysis using the Mann-Whitney U test to assess the significance of the differences in F1-score distributions across prompting strategies. The results are summarized in Table 5. We can observe that the Raw-inst strategy consistently outperformed both Few-shot and Auto-CoT prompting in terms of F1-score. Interestingly, Auto-CoT did not show a statistically significant advantage over Few-shot prompting. These findings underscore the robustness of our results and provide compelling evidence that Raw-inst prompting is the most effective strategy among those evaluated in our study.

Table 6: Top-5 Performing LLMs on the SecReq Dataset

Model	Strategy	Precision	Recall	F1-Score
deepseek-r1	Raw_inst	**0.82**	0.84	**0.83**
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	0.79
gemma2	Raw_inst	0.65	0.94	0.77

> **Finding 2:** The raw-inst prompting strategy proved to be the most effective, with every model showing an increase in F1-score and an average F1-score improvement of +0.10 compared to our baseline prompting strategy.

In this section, we presented the aggregated results across all datasets to offer a holistic view of model behavior and the effectiveness of prompting strategies. However, the independent evaluation results for each dataset are available in our supplementary material [25]. Notably, we observed that model performance was consistently lower on the *Promise+* dataset compared to the others. We also used the *Promise_exp* [20], an earlier version of *Promise+* where the models demonstrated significantly improved performance. This discrepancy may be due to the nature of requirement specifications in *Promise+* that may be impeding the models' ability to generalize.

This observation suggests an important future direction: *conducting a qualitative, in-depth analysis of the requirements across all three datasets*. Such an analysis would help to: (1) uncover patterns or linguistic structures that make certain requirements more difficult for LLMs to classify, (2) assess whether annotation guidelines were consistently applied, and (3) evaluate the presence of ambiguities or noise that could affect model accuracy. Identifying these factors would not only explain performance disparities but also inform the development of more robust datasets and prompting techniques.

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

Deepseek-r1 with the raw-inst technique, which proved to be the most effective on the SecReq dataset, was defined as our baseline (see Table 6). We compare the performance of the baseline model to two studies [15, 19] that utilized state-of-the-art ML-based models for the same task.

Table 7 presents a comparative analysis of the precision, recall, and F1-score metrics from Studies 1 [15] and 2 [19] alongside the results of our study. For Studies 1 and 2, the table includes performance metrics under two distinct evaluation scenarios:

- **Intra-domain.** The models were trained and tested within the same specification domain, providing insights into their performance in a controlled and consistent context.

Table 7: Comparison of LLM-Based Approach with Prior ML-Based Approaches [15, 19] on Intra-Domain and Cross-Domain Evaluation for the SecRec Dataset.

Dataset	Study	Precision	Recall	F1-score
ePurse	Ours	0.97	0.81	**0.88**
	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	0.76	0.90	0.83
	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	0.81	0.74
	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

- **Cross-domain.** The models were tested on data from different domains, showcasing their generalizability. Both the best-case and worst-case performance 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 intra-domain or cross-domain factors influence the performance of the classification models.

Study 1, conducted by Knauss et al. [15], 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 7). 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 [19], 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 7). 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.

Study 1 achieved high scores when models were trained and tested on the same dataset, outperforming our approach in all specifications, except for ePurse. The differences in F1-scores were small for the GP specification (0.03), but considerable for the CPN specification (0.22). 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 0.30 for the ePurse specification, 0.25 for the GP specification, and 0.34 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 0.06 for the ePurse specification, 0.09 for the GP specification, and 0.01 for the CPN specification. In the cross-domain evaluation scenario, considering the best-case scenario, our model had the same F1-score for the ePurse specification (0.88), but higher F1-scores for the GP specification (0.83 compared to 0.61) and for the CPN specification (0.74 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, surpassing traditional approaches in both performance 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-based models. Furthermore, ML-based models are prone to overfitting training data, which can lead to a steep decline in performance when models encounter datasets with varying structures or terminology. These limitations reinforcing the advantages of LLMs highlighting their robustness to domain shifts.

> **Finding 3:** LLMs offer a practical and high-performing alternative to traditional ML-based models for classifying security-related requirements. Unlike ML-based models that require dataset-specific retraining and expert labeling, LLMs generalize effectively across domains using prompt-based strategies. In our experiments, LLMs outperformed previous state-of-the-art models in F1-score across multiple datasets and configurations.

5 Threats to Validity

This section highlights potential threats to validity and reflects on the measures taken to mitigate them.

Internal validity. Internal validity relates to whether the results accurately reflect the performance of the evaluated models, independent of external influences. The study relied on three datasets, PROMISE+, SecReq, and DOSSPRE. We knowledge that limitations in the data quality and labeling could have introduced biases that

affected the evaluation results. To mitigate such a 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. 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. 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 a particular subset of software requirements, with a special emphasis on security-related ones. While the selected 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. However, we offer a reproducible framework, so that researchers can extend our study to other types of requirements.

Construct validity. Construct validity focuses on whether the evaluation approaches and metrics effectively measure the intended objectives. We 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 (see Section 6). 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.

6 Conclusion

This study evaluated the effectiveness of LLMs in classifying security-related software requirements, providing a detailed comparison with state-of-the-art ML-based models. To the best of our knowledge, this is the first systematic benchmark of LLMs applied to the classification of security-related requirements across three real-world datasets. This fills an important research gap, given the increasing interest in the automation of NFR analysis. Unlike traditional ML-based models, which typically require retraining and labeled data for each new dataset, LLMs demonstrated robust performance in cross-domain evaluations without fine-tuning. This significantly reduces setup costs and manual effort, making LLMs a scalable alternative for industry and research.

We systematically compared prompting techniques and found that a hybrid strategy—referred to as *raw-inst*, which combines role-based and instruction-based prompting—consistently improved performance across models. Statistical tests confirmed that these improvements are significant and not due to random variation. Across all prompting strategies, recall remained consistently strong. However, only a subset of models achieved a favorable balance with precision. *DeepSeek-r1* with the *raw-inst* strategy outperformed all others in terms of F1-score, indicating its suitability for practical scenarios where both false positives and false negatives matter.

The findings of this study open up numerous promising avenues for future research. Future research should explore a wider range of models and prompting strategies. Varying the style, structure, and granularity of prompts offers promising opportunities to further optimize LLM performance across RE tasks. Moreover, future studies could include direct comparisons with fine-tuned models such as NoRBERT [9] and PRCBERT [22], as well as contacting authors of prior work to replicate or extend baseline results on common datasets, which have demonstrated promising results in related scenarios.

Building on the findings of our study, we envision integrating LLMs into tools like Jira and GitHub Issues to classify requirements as they are added or updated, particularly in agile and DevSecOps pipelines. Since classification occurs once per update, computational overhead remains low. Based on classification results, such tools could support: intelligent task assignment (*e.g.*, routing security requirements to domain experts), prioritization and traceability of security issues, and real-time feedback for developers and project managers. Importantly, open-source models like Mistral-Nemo and DeepSeek can be deployed locally, enabling secure, cost-effective adoption in privacy-sensitive environments. Our study shows that these models, when paired with effective prompting strategies such as *raw-inst*, deliver strong performance without requiring large-scale infrastructure.

Our study contributes to the growing body of evidence that LLMs can serve as practical, scalable, and accurate alternatives to traditional ML-based approaches for RE tasks. By making our results and artifacts publicly available, we aim to encourage further research and real-world adoption of LLMs in SE workflows.

ARTIFACT AVAILABILITY

We declare that all artifacts from our study are publicly available at https://doi.org/10.5281/zenodo.17058472 [25]. Also mirrored on GitHub at https://github.com/aisepucrio/llm-security-req-classification.

ACKNOWLEDGMENTS

This research was partially funded by the Brazilian funding agencies CAPES (Grant 88881.879016/2023-01), FAPESP (Grant 2023/00811-0), FUNCAP (BP6-00241-00276.01.00/25), and the Binational Cooperation Program CAPES/COFECUB (Ma1036/24). We also acknowledge the Brazilian company Stone[6] for their financial support.

[6] https://www.stone.com.br/

REFERENCES

[1] Ana I Anton. 1997. *Goal identification and refinement in the specification of software-based information systems.* Georgia Institute of Technology.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Benjamin Clavié, Alexandru Ciceu, Frederick Naylor, Guillaume Soulié, and Thomas Brightwell. 2023. 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*. Springer, 3–17.

[4] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2006. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In *14th IEEE International Requirements Engineering Conference (RE'06)*. 39–48. https://doi.org/10.1109/RE.2006.65

[5] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated Classification of Non-Functional Requirements. *Requirements Engineering* 12, 2 (2007), 103–120. https://doi.org/10.1007/s00766-007-0045-1

[6] Alan M Davis. 1993. *Software requirements: objects, functions, and states.* Prentice-Hall, Inc.

[7] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated handling of anaphoric ambiguity in requirements: a multi-solution study. In *Proceedings of the 44th International Conference on Software Engineering*. 187–199.

[8] Martin Glinz. 2007. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)*. IEEE, 21–26.

[9] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F Tichy. 2020. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th international requirements engineering conference (RE)*. IEEE, 169–179.

[10] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* (2023).

[11] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. 2010. Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering* 15 (2010), 63–93.

[12] 1990. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990* (1990), 1–84. https://doi.org/10.1109/IEEESTD.1990.101064

[13] Prudence Kadebu, Sunil Sikka, Rajesh Kumar Tyagi, and Panashe Chiurunge. 2023. A classification approach for software requirements towards maintainable security. *Scientific African* 19 (2023), e01496.

[14] Haeng-Kon Kim and Youn-Ky Chung. 2005. 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*. Springer, 769–777.

[15] Eric Knauss, Siv Houmb, Kurt Schneider, Shareeful Islam, and Jan Jürjens. 2011. 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*. Springer, 4–18.

[16] Armin Kobilica, Mohammed Ayub, and Jameleddine Hassine. 2020. Automated Identification of Security Requirements: A Machine Learning Approach. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '20)*. Association for Computing Machinery, New York, NY, USA, 475–480. https://doi.org/10.1145/3383219.3383288

[17] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. 2023. Better zero-shot reasoning with role-play prompting. *arXiv preprint arXiv:2308.07702* (2023).

[18] Gerald Kotonya and Ian Sommerville. 1998. *Requirements engineering: processes and techniques.* Wiley Publishing.

[19] Tong Li. 2017. Identifying security requirements based on linguistic analysis and machine learning. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 388–397.

[20] Márcia Lima, Victor Valle, Estevão Costa, Fylype Lira, and Bruno Gadelha. 2019. Software Engineering Repositories: Expanding the PROMISE Database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering* (Salvador, Brazil) *(SBES '19)*. Association for Computing Machinery, New York, NY, USA, 427–436. https://doi.org/10.1145/3350768.3350776

[21] Maria-Isabel Limaylla-Lunarejo, Nelly Condori-Fernandez, and Miguel R Luaces. 2023. Towards a FAIR Dataset for non-functional requirements. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 1414–1421.

[22] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. 2022. 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*. 1–13.

[23] Lezhi Ma, Shangqing Liu, Yi Li, Xiaofei Xie, and Lei Bu. 2024. SpecGen: Automated Generation of Formal Program Specifications via Large Language Models. *arXiv preprint arXiv:2401.08807* (2024).

[24] Wei Ma, Shangqing Liu, Zhihao Lin, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, Li Li, and Yang Liu. 2023. LMs: Understanding Code Syntax and Semantics for Code Analysis. *arXiv preprint arXiv:2305.12138* (2023).

[25] Murilo Martin, Daniel Coutinho, Anderson Uchôa, and Juliana Alves Pereira. 2025. *aisepucrio/llm-security-req-classification: CBSOFT - version v2.* https://doi.org/10.5281/zenodo.17058472

[26] Daniel Mellado, Carlos Blanco, Luis E Sánchez, and Eduardo Fernández-Medina. 2010. A systematic review of security requirements engineering. *Computer Standards & Interfaces* 32, 4 (2010), 153–165.

[27] OpenAI. 2024. *GPT-4o mini: advancing cost-efficient intelligence.* Accessed: 2024-12-15.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[29] Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. 2024. A review on large Language Models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access* (2024).

[30] Reuters. [n. d.]. 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] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

[32] Kurt Schneider, Eric Knauss, Siv Houmb, Shareeful Islam, and Jan Jürjens. 2012. Enhancing security requirements engineering by organizational learning. *Requirements Engineering* 17 (2012), 35–56.

[33] Bruno Silva, Rodrigo Nascimento, Luis Rivero, Geraldo Braz, Rodrigo Santos, Luiz Martins, and Davi Viana. 2024. Promise+: expandindo a base de dados de requisitos de software Promise exp. *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (2024), 291–301.

[34] Ian Sommerville. 2013. *Engenharia de Software.* Pearson.

[35] Giriprasad Sridhara, Sourav Mazumdar, et al. 2023. Chatgpt: A study on its utility for ubiquitous software engineering tasks. *arXiv preprint arXiv:2305.16837* (2023).

[36] Yawen Wang, Lin Shi, Mingyang Li, Qing Wang, and Yun Yang. 2020. A deep context-wise method for coreference detection in natural language requirements. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 180–191.

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[38] Danning Xie, Byungwoo Yoo, Nan Jiang, Mijung Kim, Lin Tan, Xiangyu Zhang, and Judy S Lee. 2023. Impact of large language models on generating software specifications. *arXiv preprint arXiv:2306.03324* (2023).

[39] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* (2022).

[40] Zhengping Zhou, Lezhi Li, Xinxi Chen, and Andy Li. 2023. Mini-Giants:" Small" Language Models and Open Source Win-Win. *arXiv preprint arXiv:2307.08189* (2023).