Automatic Prompt Engineering: the Case of Requirements Classification

Mohammad Amin Zadenoori[1] Liping Zhao[2], Waad Alhoshan[3], and Alessio Ferrari[1]

[1] Consiglio Nazionale delle Ricerche ISTI "A. Faedo" (CNR-ISTI), Italy
mohammadamin.zadenoori@isti.cnr.it, alessio.ferrari@isti.cnr.it
[2] University of Manchester, UK liping.zhao@manchester.ac.uk
[3] Al-Imam Muhammad ibn Saud Islamic University, Saudi Arabia
wmaboud@imamu.edu.sa

Abstract. *Context and motivation:* Large language models (LLMs) are increasingly used to address requirements engineering (RE) tasks, including trace-link recovery, legal compliance, model generation, and others. *Question/problem:* Most of the existing studies rely on static, non-adaptive prompting strategies that do not fully harness the models' capabilities. Specifically, these studies overlook the potential of automatic prompting engineering (APE), a technique that allows LLMs to self-generate and fine-tune prompts to improve task performance. *Principal ideas/results:* This research preview aims to study the effectiveness of APE techniques in LLM-powered RE tasks. As a preliminary step, we perform a benchmarking study in which we compare APE techniques with more traditional prompting solutions for the task of requirements classification. Our results show that, on average and with some exceptions, APE outperforms the baselines. We outline research avenues, including the evaluation and tailoring of APE for other RE tasks, and considering the human-in-the-loop. *Contribution:* To the best of our knowledge, this is the first study to introduce APE in RE, paving the way for a deeper exploration of LLMs' potential in this field.

Keywords: prompt engineering · requirements engineering · large language models · natural language processing · LLM · NLP.

1 Introduction

Prompt engineering is the process of designing the optimal prompt to instruct a large language model (LLM) to perform specific tasks [11]. Several studies in requirements engineering (RE) are currently exploring the use of LLMs to address various problems, including model generation [3], trace-link recovery [6], and regulatory compliance [7]. However, in these studies, the definition of the prompts is typically manual, and the search for the most effective prompt for the task at hand is often based on non-systematic trial and error.

Automatic prompt engineering (APE) is a recent approach introduced to avoid manual prompt engineering and enable an LLM to self-refine its prompt

with the goal of improving its performance [11, 8]. With APE, the LLM is instructed to refine an initial prompt, possibly extended with a set of examples ("demonstrations") of correct input/output pairs, according to the few-shot prompting paradigm. This process can be iterated to steer the prompt refinement based on novel examples—e.g., those that resulted in incorrect output in the previous iteration.

In this research preview, we aim to explore the capability of APE, by adapting this strategy to the RE domain. We start by focusing on requirements classification, which is one of the most commonly addressed tasks in the RE literature [10]. We compare APE with different prompting strategies, including zero-shot, few-shot, and variants thereof, using Meta Llama 3 as a LLM. We chose this model as a starting point because it is open-source and, unlike ChatGPT, does not require an external server. This helps preserve the confidentiality of the data, as typically required by companies [4].

Our results show that APE reaches F1 and F2 of ∼80%, improving the performance by 5% in terms of F1 score and by 9% in terms of F2, with respect to the second-best performing strategies. Based on these encouraging results, we plan to further explore the potential of APE on other classic RE tasks, such as ambiguity detection and trace-link recovery. Furthermore, we plan to compare different emerging APE solutions that are currently under study in the natural language processing (NLP) field, and check under which conditions (e.g., LLM type, RE task) these achieve the best performance. We also plan to explore how APE can be combined with the human-in-the-loop to address complex generative RE tasks for which there is no ground truth, but qualitative feedback is needed from an expert, e.g., in model generation [3]. Our final goal is to devise a flexible tool specialised for RE that can be used as a starting point for any study that aims to systematically perform prompt engineering. We share our replication package at [9].

2 Automatic Prompt Engineering (APE)

Prompt engineering is the process of designing and refining prompts—i.e., textual inputs or queries given to LLMs—to improve the performance of the model on specific tasks. According to Ye et al. [8], the problem of prompt engineering can be formalised as an optimisation problem. Specifically, the goal of prompt engineering is finding the prompt p^* that achieves the best performance on a dataset D, when used as input for an LLM $\mathcal{M}$. Given a certain task, e.g., requirements classification, the dataset $D = (X, Y)$ is composed of input-output pairs. Specifically, it is composed of vectors X and Y, where each element $x \in X$ is an input and each element $y \in Y$is the corresponding ground truth output. For example, x represents the text of a requirement, and y represents the associated class. More formally, the prompt engineering problem can be specified as follows:

$$p^* = \arg\max_p \ f\left(\mathcal{M}(X, p), Y\right)$$

- $D = (X, Y)$ is the dataset that includes input and expected output pairs (x, y);
- $\mathcal{M}$ is the LLM, which can be regarded as a function that, given a prompt p and a set of input $x \in X$, produces a set of output $y' \in Y'$. Given the input x, this output y' is expected to match the corresponding y from the pair (x, y) of D;
- f is the evaluation function to be maximised, e.g., precision, recall, F1 score, or other metrics, depending on the task addressed by $\mathcal{M}$.

When implementing prompt engineering in practice, it is appropriate to divide the dataset D into two parts, D_{train} and D_{test}. The former is used to find the optimal prompt p^*, and the latter is used to evaluate the performance of $\mathcal{M}$. Similar to supervised machine learning, this is useful to prevent overfitting, as a prompt may be optimal for D but may not perform well on novel data.

The problem of prompt engineering formalised above can be performed manually or automatically. In the first case, human experts manually craft prompts based on their domain knowledge and intuition. In the second case, we speak about automatic prompt engineering (APE). This approach includes several different proposals [11], e.g., prompt generation, prompt scoring, and prompt paraphrasing, and consists of leveraging automated tools to produce prompt alternatives and search optimal ones. In the APE solution proposed by Zhou et al. [11], and later refined by Ye et al. [8], LLMs themselves are used to produce the prompt. Specifically, the LLM is first given an *optimisation prompt* and an *initial prompt*. The optimisation prompt instructs the LLM to produce one or more modifications of an initial prompt, and provide guidance to do so. Then, the generated prompts are evaluated on D_{train}, the best prompt is selected after some iterations, and finally tested on D_{test}.

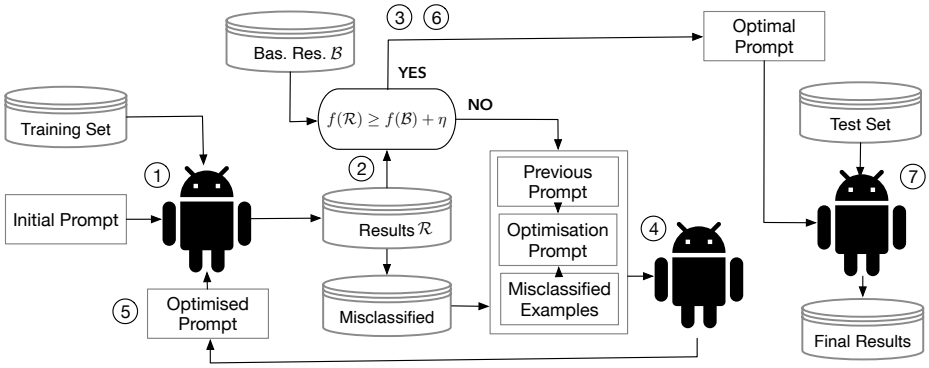

Fig. 1. APE Process Overview. Numbers refer to the text.

In our implementation, we adapt the solution by Ye et al. [8] and implement the algorithm in depicted in Figure 1. We start with an initial hand-crafted

prompt, which is evaluated on the training set ①. The results $\mathcal{R} = (\mathcal{M}(X, p), Y)$ are compared with those of a baseline $\mathcal{B} = (\mathcal{M}(X, p'), Y)$, based on the evaluation function f ②. If the baseline is outperformed by a margin η, then we have reached the optimal prompt ③. Otherwise, the LLM is prompted to optimise the initial prompt ④. To this end, it is given the previous prompt, the optimisation prompt, and a series of n examples that are selected from the misclassified cases. Based on that, the LLM generates a novel prompt (optimised prompt), which incorporates the new examples, and rephrases the previous prompt ⑤. This is fed to the LLM, and the cycle is repeated until an optimal prompt is found ⑥, or a limit of iterations l has been reached (not shown in the figure). At this point, the optimal prompt is tested on D_{test}, and the final results are produced.

The original solution [8] was designed for a model that can retain the memory of the interaction—or can simulate memory retention—in their case GPT-4 [1]. Instead, our solution assumes a stateless LLM. In other terms, at each prompt, the LLM does not keep track of the previous interactions, so the optimisation process is all *external* to the LLM. In a scenario in which one retains the memory of the interactions, the results do not depend solely on the final prompt generated but on all the incremental information fed to and retained by the LLM. This makes their results less replicable compared to our solution.

In our implementation, we start with an initial prompt that is based chain-of-thought (CoT) and is augmented with examples, as in few-shot prompting. The template for the initial prompt is the following one.

Initial Prompt

You are an expert system that needs to classify software requirements into two exclusive categories: `{A}` and `{B}`
Let's analyze the classification of requirements step by step.
Step 1: Review the examples of different types of requirements and their classifications:

- `{Example}` → `{A}`
- `{Example}` → `{B}`
- ...

Step 2: Read the explanations for these classifications:

- `{Explanation A}`
- `{Explanation B}`

Step 3: Understand how to classify requirements using the examples and explanations as guidance.
Step 4: Apply this understanding to the following requirement.
Step 5: Determine the classification of the requirement and provide the final label of the class without any explanations.
Based on these examples and explanations classify unseen software requirements into `{A}` or `{B}`. Just give the final label without any explanations. The output categories should be exactly the same as the categories mentioned here.

Requirement: `{Requirement Text}`

The optimisation prompt asks the LLM to incorporate misclassified examples in the original prompt. At the same time, it asks the LLM to rephrase the prompt and improve it, to better characterise the task based on the novel examples. Our optimisation prompt is reported below.

Optimization Prompt

You are required to enhance and clarify the explanations of the categories in the prompt by integrating illustrative examples and information implicitly referenced in the initial context. The optimized prompt must follow these strict guidelines:
Maintain the Original Steps: The steps in the optimized prompt must remain exactly the same as in the sample prompt; no changes should be made to the steps' structure or order.
Expand Explanations: Enrich and expand the explanations of each category within the steps, incorporating examples provided. Use these examples to enhance understanding and provide clarity, but ensure all content remains within the existing steps and does not extend beyond them.
Incorporate Class Explanations: Specifically, integrate the detailed "Class Explanations" of categories from the first prompt into the optimized prompt. For each category, introduce implicit clarifications based on relevant data extracted from the context, keeping all additions within the boundaries of the original steps.
End Strictly After Step 5: The optimized prompt must strictly end after step 5. Do not add any additional steps, conclusions, or content beyond this point.

3 Preliminary Experiments

Our overarching research goal is to evaluate APE techniques to address RE tasks. In this work, we focus on requirements classification. We thus formulate the following research question (RQ): *What is the effectiveness of APE for requirements classification?* To answer the RQ, we use the requirements classification dataset from Dalpiaz *et al.* [2]. We implement APE as illustrated in Section 2, considering the Meta-Llama-3-8B-Instruct LLM[4], and we compare the performance with a set of prompt baselines (cf. *Prompt Baselines* below). We chose this version of Llama since it is fine-tuned to follow human instructions. We set the temperature of the LLM to 0, to maximise output consistency. Furthermore, we set the number of examples to $n = 8$, and the expected margin over the baseline performance in terms of F1 to $\eta = 0.2$. The baseline performance is established based on the F1 of the best-performing static prompt baselines. These settings were selected based on preliminary experiments. We set the maximum number of iterations $l = 20$, and we vary the size of D_{train} considering a 10%, 20% and 30% portion of D. In addition, we consider two APE versions: one in which only the examples are changed, while the prompt is fixed (referred to as APE-fixed in the following); and another one in which both change. These choices allow us to explore different configurations while limiting resource consumption.

Task The framework by [2], distinguish requirements aspects into *Functional (F)*: a requirement includes either a functional goal or a functional constraint; and *Quality (Q)*: a requirement includes a quality goal or a quality constraint. A requirement can possess only F aspects, only Q aspects, both aspects (F+Q), or none (information). Following [2], we thus aim to solve four binary problems:

[4] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct.

- **F:** does a requirement possess functional aspects?
- **Q:** does requirement possess quality aspects?
- **OnlyF:** does a requirement possess only functional aspects?
- **OnlyQ:** does a requirement possess only quality aspects?

Prompt Baselines The following prompt baselines have been considered:

Zero-shot The model performs a task without any prior examples.
Few-shot The model is given a few examples to better understand the task.
Chain-of-Thought (CoT) The model is guided through a step-by-step reasoning process that break complex tasks into smaller, logical steps.
CoT ∪ Few-shot Combination of CoT and Few-shot.

Table 1. Results of APE-fixed and APE compared to the baselines. Cases in which the baselines outperform APE indicate that the maximum number of iterations has been reached.

	Average				**F**		**Q**		**onlyF**		**onlyQ**	
	P	**R**	**F1**	**F2**	**F1**	**F2**	**F1**	**F2**	**F1**	**F2**	**F1**	**F2**
Zero-shot	0.58	0.58	0.51	0.54	0.76	0.87	0.74	**0.86**	0.02	0.01	0.54	0.44
Few-shot	0.82	0.62	0.62	0.61	0.84	0.89	0.47	0.35	0.8	0.91	0.38	0.29
CoT	0.82	0.7	0.74	0.71	0.85	0.89	0.68	0.6	0.78	0.82	0.63	0.54
CoT ∪ Few-shot	**0.87**	0.67	0.71	0.68	0.85	**0.91**	0.55	0.44	**0.86**	**0.93**	0.57	0.48
APE-fixed (10)	0.82	0.78	0.78	0.78	0.85	0.87	0.78	0.75	0.8	0.87	0.68	0.62
APE-fixed (20)	0.81	0.78	0.78	0.78	0.85	0.88	0.78	0.78	0.81	0.86	0.67	0.61
APE-fixed (30)	0.82	0.76	0.76	0.77	0.86	0.89	0.71	0.78	0.79	0.8	0.68	0.61
APE (10)	0.79	0.78	0.76	0.76	0.84	0.85	0.73	0.7	0.8	0.87	0.68	0.62
APE (20)	0.8	0.77	0.76	0.76	0.86	0.83	0.72	0.72	0.81	0.86	0.67	0.61
APE (30)	0.79	**0.82**	**0.79**	**0.8**	**0.88**	0.87	**0.79**	0.79	0.83	0.88	**0.7**	**0.63**

Results and Discussion Table 1 reports the results in terms of F1 and F2 scores. Precision and recall have not been reported for single classes due to space limitations. We see that, on average, *all* the APE solutions, regardless of the size of D_{train} and the choice of fixed *vs* variable prompt, outperform the best baselines on F1 (0.79 *vs* 0.74, 5%) and F2 (0.8 *vs* 0.71, 9%). The best-performing strategy is APE (30), so with variable prompts and 30% of D as D_{train}, suggesting that a large set of training samples and higher prompt variability are more effective. On precision, however, CoT ∪ Few-shot is more accurate. Furthermore, for specific classes, Zero-shot and CoT ∪ Few-shot show better performance, especially on F2, which favours recall over precision. These results suggest that the APE solution may be more general, i.e., it can address different classes better, but for specific classes different solutions are preferable. This indicates that existing manual prompt engineering approaches are not ruled out, but need to be evaluated together with APE to achieve optimal performance.

It should be noted that our results do not outperform existing solutions, see e.g., Hey et al. [5]. However, APE is using a maximum of 30% of the data, which is a smaller percentage with respect to the 70% to 90% used for training in Hey et al. Furthermore, our goal is to see whether APE improves over static prompt engineering solutions rather than outperforming other baselines.

Threats to Validity Given the preliminary nature of the study, several factors can affect its outcomes. There is a possible inconsistency of the output in multiple runs. To minimise this, we set the LLM temperature to 0. The performance also depends on the initial prompt, which is rephrased in the iterations. Hand-crafted prompt engineering is thus not entirely ruled out. Future works will address this issue, e.g., by providing approaches to design effective initial prompts.

4 Research Plan, Risks, and Mitigations

The following RQs will be addressed as part of our future research:

RQ1 *What is the effectiveness of APE for RE tasks?* To address this RQ, we first plan to survey the literature to identify the most promising APE solutions to be adapted to the RE field. Based on these approaches, we plan to perform a set of benchmarking studies on the classical discriminative (i.e., non-generative) RE tasks, such as ambiguity detection, and trace-link recovery. This will guide in the selection of RE-specific APE solutions that perform best on specific tasks, and will lead to a catalogue of task-specific APE templates. We will also study the characteristics of the optimal prompt identified and why, in certain cases, the fixed baselines still outperform APE.

RQ2 *How to combine APE with human-in-the-loop for RE tasks?* Discriminative RE tasks are typically associated with a ground truth that can be used as a training set. In these tasks, a quantitative evaluation is typically possible, and this can drive prompt optimisation. However, some generative and more creative RE tasks, such as model synthesis and idea generation, require a qualitative evaluation [3]. This judgment can, in principle, be again automated by LLMs, e.g., by prompting them to express opinions on the produced output. However, as humans are the ultimate users of the output, they should enter into the decision process. Furthermore, the ground truth may not exist also in discriminative tasks, and humans may need to provide feedback on the output proposed in each iteration. To understand how to harness the collaboration between LLM and humans, we plan to perform controlled experiments comparing LLM-intensive and human-intensive solutions in complex generative tasks such as model synthesis, on which we have acquired previous experience [3].

RQ3 *How to devise a tool that supports APE for RE?* The knowledge acquired in RQ1 and RQ2 will be exploited to implement a holistic tool for APE in RE that can be applied to multiple tasks. The base skeleton of the tool is already part of our prototype in [9]. We will then use design science to implement and refine the tool's interface based on incremental evaluations with users. To this end, we will perform usability tests of the tool and collect task completion time, error rates, and user satisfaction.

RQ4 *Does APE enhance the job of requirements engineers?* APE can be resource-consuming, and, in case of limited computational power, responses to queries can take time—with a training set of $\approx$ 200 requirements, each optimization cycle takes $\approx$20 minutes on Intel Core i9-13900K 128 GB 4 x 32 GB DDR5 6000 MT/s with GeForce RTX 4090 OC Edition 24GB GDDR6X. While

this is acceptable for tasks that can be performed in batch, such as classification, human-in-the-loop contexts require shorter response times. It is therefore important to devise efficient APE solutions (e.g., by reducing the training set or the number of iterations) that are acceptable to requirements engineers, and that can make their job smoother rather than more frustrating. In addition, it is important to understand whether APE really improves over manually crafted prompts in practical scenarios, or if requirements engineers are more efficient and effective with manual prompting. To explore these issues, the tool developed in RQ3 should undergo a field experiment, i.e., an experiment in a real-world setting. Given our connection with railway companies and our expertise in ambiguity detection, we plan to focus our field experiment on this task and on this domain.

Risks and Mitigations The main risk to be addressed is the resource consumption of APE, which can make multiple iterations demanding in terms of time and power. Another issue is the proliferation of APE solutions and LLMs, which could make our tool quickly outdated. To mitigate this, our initial design has been defined to accommodate different, possibly novel, models, and it is not specific to the given model. Furthermore, since the tool is planned to be open source, novel APE approaches and LLMs can be incorporated by their users.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Dalpiaz, F., Dell'Anna, D., Aydemir, F.B., Çevikol, S.: Requirements classification with interpretable machine learning and dependency parsing. In: RE'19. pp. 142–152. IEEE (2019)
3. Ferrari, A., Abualhaijal, S., Arora, C.: Model generation with llms: From requirements to uml sequence diagrams. In: REW'24). pp. 291–300. IEEE (2024)
4. Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., Gnesi, S., et al.: Natural language requirements processing: a 4D vision. IEEE Software **34**(6), 28–35 (2017)
5. Hey, T., Keim, J., Koziolek, A., Tichy, W.F.: NoRBERT: Transfer learning for requirements classification. In: RE'20. pp. 169–179. IEEE (2020)
6. Rodriguez, A.D., Dearstyne, K.R., Cleland-Huang, J.: Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In: REW'23. pp. 455–464. IEEE (2023)
7. Santos, S., Breaux, T.D., Norton, T.B., Haghighi, S., Ghanavati, S.: Requirements satisfiability with in-context learning. In: RE'24. pp. 168–179. IEEE (2024)
8. Ye, Q., Axmed, M., Pryzant, R., Khani, F.: Prompt engineering a prompt engineer (2024), https://arxiv.org/abs/2311.05661
9. Zadenoori, A.: Automated Prompt Engineering: the Case of Requirements Classification (Replication Package) (2024). https://doi.org/10.5281/zenodo.14005656
10. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. ACM Computing Surveys (CSUR) **54**(3), 1–41 (2021)
11. Zhou, Y., Muresanu, A.I., Han, Z., Paster, K., Pitis, S., Chan, H., Ba, J.: Large language models are human-level prompt engineers (2023), https://arxiv.org/abs/2211.01910