Unveiling Power on Combining Prompt Engineering Techniques: An Experimental Evaluation on Code Generation

Cristofer Girardi[1], **Damires Yluska de Souza Fernandes**[1], **Alex Sandro da Cunha Rêgo**[1]

[1]Federal Institute of Paraiba - IFPB
João Pessoa – PB – Brazil

cristofer.girardi@academico.ifpb.edu.br, {alex,damires}@ifpb.edu.br

Abstract. *Prompt engineering techniques have seen a significant rise in research interest as a means to achieve satisfactory results without retraining Language Models. This work presents a set of experiments to analyze the power of a combination of prompts. To this end, it evaluates six prompt techniques, combining them to result in twelve experimental scenarios applied to Python code generation. Evaluation using BERTScore indicates that Role combined with RAG achieves the highest performance in code generation with 98% similarity. Skeleton-of-Thought and Self-Verification reveal promising opportunities for the design of prompt templates. Our findings contribute to unveiling the power of combining prompt techniques for current applications such as code generation.*

1. Introduction

Artificial Intelligence (AI) has shown remarkable notoriety after OpenAI's ChatGPT made evident the potential of Large Language Models (LLMs) [Korzyński et al. 2023]. Likewise, other companies have also introduced competitive generative models, such as the Llama, Claude or Gemini. With the growing demand for more accurate results, Prompt Engineering (PE) has emerged as an important practice in the context of LLMs to achieve satisfactory results, i.e., solutions that meet the defined requirements, without retraining them [Vatsal and Dubey 2024]. According to [Schulhoff et al. 2025, Medeiros et al. 2024], PE is an iterative process that involves developing effective prompts through the use of diverse techniques. A prompt technique encompasses the development of structured instruction sequences that take advantage of a given LLM in various applications and research domains. To the best of our knowledge, currently, there are 33 terminologies of prompting, 58 text-only prompting techniques, and 40 ways applicable to other modalities like image and video [Schulhoff et al. 2025].

In this sense, a user utilizes an LLM-based application such as ChatGPT, formulating prompts that are readily interpretable by the model to generate a determined response. [Sabit 2023] designed tips, best practices, iterative refinement, balance of user intents, harnessing external resources, and the usage of ethical guidelines to enhance the responses of a given LLM. For instance, tools such as Copilot and Gemini enhance code generation when developers formulate more structured and contextually appropriate prompts [Mahir et al. 2024]. Additionally, [Neves et al. 2024] demonstrated that the use of prompts plays a critical role in shaping LLM performance, showing that carefully designed prompting can significantly influence outcomes and enhance competitiveness, particularly in applications such as sentiment analysis.

The prompt design depends on several factors. According to [Bansal 2024], it is necessary to tailor LLM through PE to achieve a specific goal, ensuring that the outputs are relevant and valuable while minimizing issues, such as hallucinations and non-determinism. This can be achieved using simple PE techniques — such as Zero-Shot, Few-Shot, Chain-of-Thought, Role Prompt, Retrieval Augmented Generation (RAG) — as well as more refined strategies like Skeleton-of-Thought; Based on the analysis of various studies on PE techniques and their applications [Vatsal and Dubey 2024, Shin et al. 2023, Reynolds and McDonell 2021, Ning et al. 2024] it is possible to see that some techniques have been more addressed in the literature while others, although with potential, not so much. Then, the combination of simple and/or more refined PE techniques is still an open field for research and experimentation. Developing frameworks that combine these techniques may enhance some evaluation approaches, such as code generation. This view is supported by [Neves et al. 2024], who emphasize the need to explore prompt combinations that could enhance LLM performance.

In this scenario, we define the research questions of the present work as follows: "*Can the combination of prompt engineering techniques bring more effective results when using LLMs? Do some techniques that have not yet been much studied, have potential for more accurate results on such combination?* ".

We have accomplished an experimental evaluation in order to answer our research questions. To this end, we have chosen some simple and more refined PE techniques and some possible combinations of them in order to evaluate some reputable LLMs. As a use case, we have defined Python code generation as the problem to be solved and studied on two competitive LLMs (GPT-4o-mini and Llama-3.1-405b).

When using a given LLM for code generation, some challenges need to be considered, such as syntactic and semantic aspects and the efficiency of the generated code itself. The aim is to include sufficient contexts and instructions, by means of a PE technique, so that the LLM can reason and provide a response that meets the mentioned criteria [Coignion et al. 2024, Sarker et al. 2024]. While all of the criteria mentioned are important, in this use case, we consider that an LLM meets the requirements by measuring the similarity of the generated code w.r.t. its ground truth.

Thus, the overall idea of this work is to introduce a novel perspective for the combination of PE techniques, particularly to some combining possibilities that have not been yet much studied or experimented with. Hence, our contributions are as follows: (i) We provide definitions of combinations of simple used PE techniques with others sparsely investigated; (ii) we accomplish an experimental evaluation regarding the combined scenarios applied to code generation; (iii) we highlight some opening new perspectives of templates of PE techniques and other ones less mentioned.

The paper is organized as follows: Section 2 introduces some theoretical background. Section 3 discusses some related works. Section 4 describes the experimental design. Section 5 presents the results obtained. Section 6 provides a discussion about the research findings, and Section 7 concludes the paper and indicates future works.

2. Theoretical Background

Some concepts and techniques w.r.t. the core of our study are introduced in the following.

2.1. Prompt Engineering Techniques

This study explores six PE techniques, which are described as follows.

- **Role Prompting** is a technique used to guide the LLM to approach a question or problem by assuming a specific role or character, i.e., a persona [Schulhoff et al. 2025]. The model output becomes more reliable when an appropriate role is assigned to the LLM at hand [Reynolds and McDonell 2021].
- **Rephrase and Respond** (RaR) is a PE technique whose user's instruction is analyzed by the LLM in terms of the sentence, adapting or even rewriting it [Schulhoff et al. 2025]. This means that, before generating an answer for the user-defined prompt, the model engages in sequential reasoning by creating a new sentence and applying it again to itself in order to improve the output. This task is accomplished by appending at the end of the user query the term "*Rephrase and expand the question, and respond*", which enables the model to enhance its response generation process [Deng et al. 2023].
- **Zero-Shot-CoT** (ZS-CoT) is a concatenation of Zero-Shot (ZS) and Chain-of-Thoughts (CoT) techniques. To help matters, they are explained individually: ZS is a technique that does not rely on examples to provide additional context to LLM [Schulhoff et al. 2025]; on the other hand, CoT works as an interaction of prompts to achieve an answer or a creation of logical reasoning to give a context to LLM [Wei et al. 2022]. The integration of these two techniques is designated as Zero-Shot-CoT. It refers to creating a task-agnostic prompt that eliminates the need of interaction or answer-extraction by means of explicitly adding a sentence at the end of the user's query like "*Let's think step by step*", "*First, let's think about this logically*" [Kojima et al. 2022] or "*Let's work this out in a step-by-step way to be sure we have the right answer*" [Zhou et al. 2022].
- **Skeleton-of-Thoughts** (SoT) is a technique that decomposes the user's question or instruction into parallel lines of reasoning, generating concurrent responses. These outputs are then concatenated to synthesize the final answer [Ning et al. 2024].
- **Self-Verification** (SV) is an automated validation technique used to enhance the reliability of LLM outputs. By generating multiple candidate responses and assigning them to verification scores, the model identifies the most reliable answer based on predefined metrics such as BLEU, ROUGE-n and ROUGE-L, and BertScore, ensuring higher accuracy before delivering the result to the end user [Weng et al. 2022].
- **Retrieval Augmented Generation (RAG)** is a technique model that leverages external data sources (e.g., Wikipedia) not originally included in the LLM's training to retrieve relevant document snippets, which are then combined with the main input prompt as additional context [Lewis et al. 2020]. This process involves three steps: (i) segmenting the documents or data into chunks, (ii) encoding these chunks, and (iii) indexing and storing them in a vector database. When a user submits a query, it is encoded, matched against the index to retrieve the top-K nearest documents, which are concatenated with the main prompt to provide enriched context for the LLM's response.

2.2. Evaluation Metrics

There are many metrics to evaluate LLM responses. Particularly, in this work, we use the BertScore [Hu and Zhou 2024] measure, which leverages BERT [Devlin 2018] pre-trained embeddings to assess semantic similarity w.r.t. the meaning and correspondence of LLM-generated outputs in comparison to reference texts. The underlying rationale of the BertScore regards some sequential operations as follows:

1. **BERT embedding loading**: the pre-trained BERT model is utilized to obtain contextualized token embeddings;
2. **Token representation**: for each token in the generated text $\{x_i\}_{i=1}^{N}$ and the reference text $\{y_j\}_{j=1}^{M}$ corresponded embedding are extracted. Each token is represented as fixed length vectors denoted by $\{\hat{x}_i^e\}_{i=1}^{N}$ and $\{\hat{y}_j^e\}_{j=1}^{M}$. The e denotes the embedding space, indicating that $\hat{x}^e$ and $\hat{y}^e$ are the mean-pooled embeddings of the candidate and reference. These are used to compute BERTScore via cosine similarity.
3. **Similarity Measure**: the sequence of vectors is consolidated into a singular representation to quantify the similarity between the two texts, according to Equation 1:

Lastly, the cosine similarity function is applied to measure the similarity between the vector representations of the references and generated texts, denoted as follows (Equation 2):

$$\widehat{\mathbf{x}}^{\mathrm{e}} = \frac{1}{N}\sum_{i=1}^{N}\widehat{\mathbf{x}}_i^{\mathrm{e}}, \quad \widehat{\mathbf{y}}^{\mathrm{e}} = \frac{1}{M}\sum_{j=1}^{M}\widehat{\mathbf{y}}_j^{\mathrm{e}} \quad (1) \qquad \text{BertScore} = \frac{\hat{x}^e \cdot \hat{y}^e}{\|\hat{x}^e\| \cdot \|\hat{y}^e\|} \quad (2)$$

Thus, the result produces a similarity score within the interval [-1..1], where: 1 indicates identical vectors (high similarity); 0 represents orthogonal vectors (no similarity); -1 denotes complete dissimilarity.

3. Related works

PE techniques have been used in literature as a means for improving the outputs of LLMs. Some of them, applied to code generation, are described in the following.

[Wang et al. 2024] proposed a framework that develops an understandable approach to Least-to-Most Prompting, CoT, and Few-Shot techniques to enhance Python code generation. Their study designed and evaluated prompt strategies—including one-shot examples, dynamic examples, general guides, multi-step conversational prompts, and all-in-one prompts—to assess their impact on LLM performance. Experiments were conducted using GPT-4, GPT-4o, Llama3-8b, and Mixtral-8x7b across 120 programming questions (100 from LeetCode[2] and 20 from USACO[3]). The study underscored the importance of tailored PE strategies in improving LLM reasoning, code correctness, and efficiency, offering structured guidelines for optimizing LLM-based code generation tasks. Performance was evaluated using three key metrics: Pass Rate (solutions correctness),

[2]https://leetcode.com/
[3]https://usaco.org/

Time Spent (coding efficiency), and Pylint score (code quality). Their study suggested that GPT-4o, using the "multi" prompt strategy, showed superior ability to understand and generate accurate responses.

For [Damke et al. 2024], the use of prompt engineering is crucial to achieving good results in code generation. In this study, rounds of running prompt engineering techniques (Zero-Shot, Few-Shot and CoT) were evaluated using the Pass@K metric. This metric evaluates the probability that at least one of K generated code sample solves a problem. To reduce variance, the paper uses an unbiased estimator by generating $n \geq k$ samples, counting the correct ones c, and computing pass@k according to [Chen et al. 2021]. A small dataset was created with 12 Python programming exercises of varying difficulty levels. Three PE techniques - Zero-Shot, Few-shot, and Chain-of-Thought - were tested, with each one running three times. The work revealed that the Zero-Shot technique yielded better outcomes based on the pass@k metric result.

[Zheng et al. 2024] proposed a unified framework for code generation that integrates single-turn and multi-turn CoT technique, combining reasoning, instruction, and execution feedback prompts to enhance LLM performance. Using the CodeContests and TACO datasets, stratified by difficulty, they evaluated some models such as Llama 3.0, Llama 3.1, and GPT-4o with the Pass@K and Pass-n@K metrics to measure success rates under fixed sampling. Their findings show that while multi-turn prompting alone yields limited gains, combining it with CoT strategies significantly improves performance, establishing a robust baseline for understanding prompt interactions and optimization in multi-turn code generation.

Compared to these studies, our work takes a distinct approach. While prior research typically applies commonly used PE techniques, such as Few-Shot and Chain-of-Thought, either individually or in combination to generate specific responses, they often incorporate continuous interactions by iteratively providing context to improve model outputs, using evaluation metrics like pass@k or dataset-specific measures to assess generated code. However, these studies generally maintain a systematic framework that evaluates each technique in isolation. In contrast, our work introduces a novel perspective by exploring the combined use of both commonly used and refined PE techniques, aiming to advance the analysis of their synergistic effects beyond the commonly studied configurations.

4. Experimental Design

The dataset and experimental setup underlying this work are described in this section.

4.1. Dataset

The dataset used in this work has been extracted from the work of [Gouveia et al. 2023], which consists of a collection of Python programming-related questions and answers. The underlying database has been generated by means of a tool that assists students in programming competitions. For instance, those students participate in programming marathons organized by the Brazilian Computer Society (SBC). The training methodology is entirely problem-based, emphasizing hands-on practice through real coding questions.

From the original database, we extracted the necessary data for our experimental evaluation. Initially, the full dataset comprised 457 columns and 9,575 instances. Upon detailed analysis, we observed that questions and answers were duplicated across student records. To address this, we implemented a data-cleaning process to select the most relevant features and data: the unique questions, student submissions, and the corresponding ground truth (i.e., the correct answers for each question). To ensure consistency, we filtered the instances to acquire only Python language questions with verified correct answers and submissions associated with a superuser, who has the correct answers. This process resulted in the construction of a ground truth dataset containing 70 curated instances, with all columns standardized as string-type data.

Thus, a set of six features has been selected as relevant for experimental evaluation of LLMs w.r.t. the combination of PE techniques, as shown in Table 1.

Table 1. Dataset Features

Column Name	Description
title	Question title
desc	Question description
input_desct	Input parameters from the question
output_desc	Expected output
dicas	Question tips
program	Correct answer

4.2. Experimental Setup

According to the research questions defined at Section 1, the primary goal of this work is to investigate the combination of some PE techniques to evaluate if these combinations may improve the performance of LLMs. Particularly, such evaluation regards whether the LLM is able to produce similar Python codes w.r.t. the ground truth.

The selected LLMs for this study are GPT-4o-mini and Llama-3.1-405b. The reasons underlying these choices refer to their comparable capabilities in terms of: (i) similar levels of model size and reasoning power, and (ii) supported input/output formats and prompt design flexibility. The criteria outlined aim to provide fairness with respect to how the models can be compared[4] within the objectives of our experimental evaluation.

To maintain a focused and efficient experimental scope, we randomly selected 10 Q&A instances from the ground truth dataset, independently of difficulty level (Basic 20%, Intermediate 40%, Advanced 40%), to ensure an unbiased evaluation. A two-iteration loop was implemented to capture variability in the outputs generated by the LLMs. For the code generation evaluation phase, BertScore was employed to assess the semantic and structural alignment between the ground truth codes and the generated responses, offering a robust measure of how closely the outputs match in both meaning and form. The experimental scenarios consisted of various prompt combinations, beginning with a baseline and extending across twelve additional combined configurations. Upon completing all runs covering 10 questions, 2 repetitions, 12 scenarios, and 2 LLMs, we obtained a total of 480 outcomes; when including the baseline results, the final count reached 520 experimental results.

Table 2 depicts the selected PE techniques and combination scenarios (S_i) chosen for our evaluation. The rationale behind the selection of the PE techniques in this study lies in analyzing commonly used techniques (P_1, P_3 and P_6) with prompts that have not

[4]Artificial Analysis Comparative

yet been the focus of significant investigation (P_2, P_4 and P_5).

Table 2. PE Techniques, Models and Experimental Scenarios

PE Techniques	Scenarios Combined	LLM models used
P_1 - Role Prompting	S_0: P_1	GPT-4o-mini
P_2 - Rephrase and Respond (RaR)	S_1: $P_1 + P_6$	Llama-3.1-405b
P_3 - Zero-Shot-CoT (ZS-CoT)	S_2: $P_1 + P_2 + P_6$	
P_4 - Skeleton-of-Thoughts (SoT)	S_3: $P_1 + P_3$	
P_5 - Self-Verification (SV)	S_4: $P_1 + P_3 + P_6$	
P_6 - Retrieval Augmented Generation (RAG)	S_5: $P_1 + P_4 + P_6$	
	S_6: $P_1 + P_5 + P_6$	
	S_7: $P_1 + P_2 + P_4 + P_6$	
	S_8: $P_1 + P_2 + P_5 + P_6$	
	S_9: $P_1 + P_3 + P_4 + P_6$	
	S_{10}: $P_1 + P_3 + P_5 + P_6$	
	S_{11}: $P_1 + P_2 + P_4 + P_5 + P_6$	
	S_{12}: $P_1 + P_3 + P_4 + P_5 + P_6$	

In the light of our experimental evaluation, we describe each defined prompt technique and the scenarios with their combinations in the following.

PE techniques

The P_1 Role prompt technique has been organized into two distinct purposes: (i) baseline (S_0): "*You are a programming tutor in Python*" was used to point out the specific coding tutor role, without inducing any other type of aid to the model's reasoning; (ii) in the rest of the combination scenarios (S_1...S_{12}), additionally to the Role prompt (coding tutor), specific instructions were added according to the type of PE technique, as explained in the following.

For P_2 (RaR), we incorporated the sentence "Rephrase and expand the question, and respond" at the end of prompts, inspired by [Deng et al. 2023]. In the P_3 (ZS-CoT), there are sentences that could be used, but we followed the state-of-the-art phrase "*Let's think step by step*" at the end of prompts, as proposed by [Kojima et al. 2022].

In P_4 (SoT), it was necessary to decompose the user's question into three distinct lines of thought, allowing the LLMs to process the code generation task in parallel and generate different responses. These outcomes were then concatenated into a single final answer. Each statement is detailed as follows:

- **The first thought** is to extract key information from each question, including the main query, any tips, required input data, the question type, the expected output format, and the intended goal. This structured extraction ensures that the LLM receives sufficient context to reason effectively and align its responses with the user's objectives.
- **The second thought** is to generate a Python code using the user's question as a reference and adding comments into the code.
- **The third thought** involves generating new code if a sample code has been provided by the user.

For P_5 (SV), the objective is to receive the response from the LLMs by performing a double-check mechanism in each return. Then, we resend the prompt three times to the

LLMs with the following instruction: "*Please rewrite the question and answer to provide a better response*", following the approach proposed by [Weng et al. 2022]. We evaluate each generated code by LLM, using the BertScore metric, with the highest-scoring response being selected as the final output.

At last, w.r.t. the P_6 (RAG), the goal is to enhance model performance by retrieving similar question-answer pairs from a vector database, where our ground-truth data was first chunked, encoded, and indexed before being stored. This additional context may allow the LLM to generate more accurate and relevant responses despite lacking prior knowledge of the ground truth data.

Combination Scenarios

The baseline (S_0) is provided with minimal guidance employing the Role Prompt technique and the user's question to assess the inherent capability of the LLMs to perform reasoning and code generation. Regarding the scenario S_1, we gather the Role and RAG techniques by incorporating the user's prompt with the response type returned by LLM. In S_2, we deal with three prompt techniques (Role, RaR, and RAG), by adding the user's prompt with the expected response type. Scenarios S_3 and S_4 employ the ZS-CoT technique with distinct approaches. S_3 followed the ZS-CoT technique, no context, only using (Role and ZS-CoT), and the user's question. However, S_4 enhances contextual understanding by integrating both (Role, ZS-CoT, and RAG).

In scenario S_5 (Role, SoT, and RAG), the SoT technique was employed, with the aim of obtaining from the model different responses based on its interpretation of the user's question. These responses are processed in parallel, concatenated and then resubmitted along with the original user question. On the other hand, in the S_6 (Role, SV, and RAG), the user's question was input into the LLM. To ensure response accuracy, a self-verification mechanism was employed, wherein the model generated and evaluated its output over three iterations. The best response was selected based on the highest BertScore. In S_7, we define the combination of techniques (Role, RaR, SoT, and RAG) in order to send the prompt to the LLMs, in a process structured into three steps: (i) send Role, RaR and User question; (ii) as SoT works in parallel, we send three thoughts with the outcome coming from the previous prompt; (iii) finally, we concatenate all responses obtained, including the RAG one, and send to LLM to give the final response.

In the S_8, the techniques used are Role, RaR, SV, and RAG. In the first step, we send Role, RAG, RaR and the user's question. In the second step, the SV prompt is used to evaluate the answers; this process is requested 3 times to get the response with the highest score using the BertScore metric.

S_9 is composed of the following techniques: Role, ZS-CoT, SoT, and RAG. In this case, the main goal was to combine ZS-CoT and SoT, sending those prompts in parallel, plus user's question. After receiving all responses, we unify them and send a new prompt with the RAG technique to give to the LLM to generate the final answer.

The proposal of S_{10} (Role, ZS-CoT, SV, and RAG) is to operate in two stages: we send Role and ZS-CoT and user's question. Subsequently, SV and RAG techniques are applied three times to get the response with the highest score using the BertScore metric.

Scenarios S_{11} and S_{12} are more robust, aiming to integrate several PE techniques and evaluate whether the results of such integration have a positive effect. If so, this could generate a new PE template (see Section 6).

The S_{11} is composed of Role, RaR, SoT, SV, and RAG techniques. This proposal was designed to operate on four distinct stages: (i) gather Role, RaR and RAG techniques along with the User Question; (ii) as the SoT works in parallel to guarantee different responses, analyze the three different thoughts sent with Role, SoT and User question; (iii) merge the LLM responses from steps 1 and 2, then consider the new request sent with the user's question to the model and return the final output; (iv) apply SV by generating three different responses and select the one with the highest BertScore.

For S_{12}, the PE techniques Role, ZS-CoT, SoT, SV, and RAG were joined to formulate the scenario. The aim was to maintain the integrity of each technique, as in S_9, while enhancing the score without increasing complexity. The process follows three distinct stages: (i) gather Role, SoT with each different thoughts, ZS-CoT and User question, sending requests in parallel to guarantee different responses; (ii) merge the LLM responses from step 1, add the RAG, then consider the new request sent with the user's question to the model and return the final output; (iii) apply SV by generating three different responses and select the one with the highest BertScore.

5. Results

Figure 2 presents the evaluation results of the combined PE techniques for the LLMs GPT-4o-mini and Llama. This visualization illustrates how each scenario, formed by different PE combinations, performed in terms of the similarity score (BertScore). Overall, the models demonstrated high similarity between the generated code and the ground truth, with all scenarios achieving a minimum BertScore of 0.940. To graphically represent our results, we applied the harmonic mean metric to approximate a representative value for each question, since two outcomes were generated per question. The harmonic mean was selected because it is less sensitive to outliers than the arithmetic mean, making it a suitable choice for deriving a central value similar to the median. Given that the measurements are expressed in thousands, we calculated the harmonic mean of standard deviation across all scenarios (sd = 0.016) to provide a unified reference value. To enhance the clarity and interpretability, we chose to start the y-axis at 0.940 to provide an expanded view of the performance differences. Both LLMs stand out in scenarios S_1, S_4, S_6 and S_{10}.

The baseline scenario S_0, which uses only the Role prompt technique, demonstrates a remarkable similarity result (GPT-4o-mini=0.978 and Llama=0.972). In S_1 (Role and RAG), both GPT-4o-mini and Llama reached their highest similarity scores — 0.984 and 0.981, respectively — surpassing the baseline performance. In S_2 (Role, RaR, and RAG), we noticed a smooth decrease in similarity in Llama (0.971), in contrast to the GPT-4o-mini slightly surpassing (0.980). Both scenarios (In S_1 and In S_2) indicate that LLMs achieve better reasoning without a large structure of prompts combined.

On scenarios S_3 and S_4, we assessed the impact of adding semantic context. S_3 (Role and ZS-CoT) resulted in a lower similarity score compared to S_4 (Role, ZS-CoT, and RAG) in GPT-4o-mini. In S_4, we added the RAG technique in order to provide a sample of the expected result. The Llama model has benefited from such enhancement, reaching a score of 0.975 and slightly surpassing the GPT-4o-mini model (0.974). Our findings suggest that Llama's performance improves with given richer semantic context.

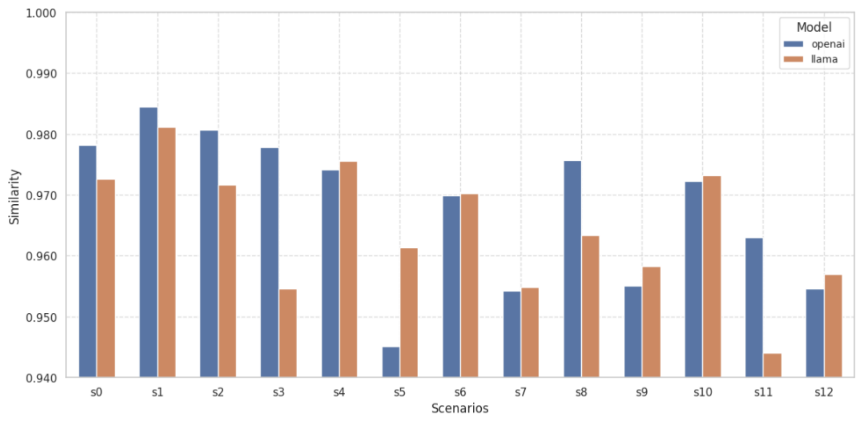

Figure 2. Similarity measurements of compared LLMs in all experimental scenarios.

On the other hand, ZS-CoT demonstrates to be more effective for the GPT-4o-mini than for the Llama model at evaluation.

Regarding S_5 (Role, SoT, and RAG), it has achieved the highest drop below the average performance for both models (Llama = 0.961 and GPT-4o-mini = 0.945). This difference can be attributed to the use of the SoT technique, which operates by sending multiple parallel requests to the API. Both APIs were operated under Free Tier conditions, meaning they ran on limited computing resources; even with these constraints, Llama outperformed GPT-4o-mini, suggesting that the SoT technique enhanced Llama's reasoning for problem-solving issues.

Concerning the S_6 (Role, SV, and RAG), the similarity measurements were practically identical between both models (Llama = 0.970 and GPT-4o-mini = 0.969). But in S_7 (Role, RaR, SoT, and RAG) and S_9 (Role, ZS-CoT, SoT and RAG), both models have a drop in measured performance, in contrast to the S_8 (Role, RaR, SV, and RAG), it shows a better performance to GPT-4o-mini (0.975) against Llama (0.963). In the scenario S_{10} (Role, ZS-CoT, SV, and RAG), the Llama (0.973) outperformed GPT-4o-mini (0.972), and surpassing its baseline. A closer examination reveals that the LLMs produced outputs with higher similarity to the ground-truth code when the SV technique was employed, effectively driving the generated code toward greater correctness. Thus, these findings suggest that incorporating the SV technique generally improves the alignment of generated code with the reference outputs compared to scenarios where it is not applied.

Finally, scenarios S_{11} and S_{12} integrated Role, SoT, SV, and RAG as core techniques, with the aim of exploring the feasibility of proposing new prompt templates (see Section 6). Whereas S_{12} employs ZS-CoT, and by incorporating RaR technique in S_{11}, we observe that in S_{11} Llama presents its lowest score (0.944), while S_{12} shows a more balanced performance for both models.

In order to determine whether the difference regarding performance measure between the LLMs over the scenarios (S_1 to S_{12}) is statistically significant w.r.t. the baseline scenario (S_0), we formulated and conducted the hypothesis tests, as outlined below.

For each pair S_0 to S_i ($i \in [1,12]$):

- $\mathbf{H_0}$: There is no significant difference between the performance of the baseline S_0

and the scenario S_i, using LLMs GPT-4o-mini or Llama ($\mu S_0 = \mu S_i$).

- **H_1**: There is significant difference in performance of LLMs GPT-4o-mini or Llama in scenario S_i w.r.t. baseline S_0 ($\mu S_0 \neq \mu S_i$).

We applied the Kolmogorov-Smirnov test [Dodge 2008] on 520 results generated by the GPT-4o-mini and Llama models to assess whether the data followed a normally distributed. The test results indicated that the measurements of both models are non-normal distributions. Thus, the Wilcoxon signed-rank test [Woolson 2005] was conducted to assess the performance across scenarios. For GPT-4o-mini, statistical differences ($p < 0.05$) were observed in scenarios S_5, S_7, S_{11}, and S_{12}, indicating deviations from baseline performance and leading us to fail to reject the null hypothesis (H_0). Conversely, scenarios S_1, S_2, S_3, S_4, S_8, and S_{10} exhibited no statistically significant differences ($p > 0.05$), suggesting performance aligned with the baseline and thus rejecting H_0.

Considering the Llama model, scenarios S_1, S_3, S_5, S_7, S_9, S_{11}, and S_{12} demonstrated statistically significant difference over S_0 ($p < 0.05$), supporting the rejection of H_0. Finally, S_2, S_4, S_6, S_8, and S_{10} did not present significant differences, and we therefore fail to reject H_0 for these cases.

6. Discussion

Throughout our investigation into which PE techniques should be employed, we observed that many researchers predominantly utilize the Chain-of-Thought technique or design new prompts that, in essence, represent combinations of existing strategies, even when such combinations are not explicitly defined [Wang et al. 2024]. A key observation is that much of the current literature emphasizes the development of new frameworks rather than fully leveraging the potential of established PE methods. As a result, researchers may overlook the fact that other techniques could effectively address many of the current challenges.

Our findings suggest that certain prompts can significantly aid in improving an LLM result. Particularly, in this work, we verified such hypothesis when applying the PE techniques in order to generate Python code. Notably, we highlight that S_1 (Role and RAG) achieved the highest obtained code similarity value, despite being the smallest PE combination, demonstrating that the LLMs have sufficient reasoning ability to generate code for specific problems without complex prompting. Similarly, S_2 (Role, RaR, and RAG) maintained excellent code similarity, despite incorporating the RaR prompt. This led us to investigate why this technique underperformed in S_8 and S_{11} when applied to Llama but not to GPT-4o-mini. Although scenarios S_{11} and S_{12} yielded lower overall results, they provided a thought-provoking perspective, illustrating the potential value of combining multiple PE techniques into reusable templates. However, designing and implementing these scenarios required careful planning, detailed examination, and precise adaptation to prevent mischaracterization or unintended interactions among the techniques.

In analyzing the scenarios S_4–S_{10}, we observed that specific combinations of prompt techniques lead to distinct code similarity outcomes depending on the LLM. This suggests that certain models may struggle to solve problems without adequate contextual structure. For example, GPT-4o-mini demonstrated superior performance in S_8 (Role, RaR, SV, and RAG) but underperformed in S_5 (Role, SoT, and RAG), where the use of

SoT introduced technical challenges, such as API request limitations and computing resource constraints. Despite having paid for API access, both GPT-4o-mini and Llama were affected by Free Tier restrictions, requiring us to throttle requests and even pause tests overnight to avoid errors. For Llama, the best-performing scenario was S_4 (Role, ZS-CoT, and RAG), whereas S_7 (Role, RaR, SoT, and RAG) produced the lowest results. Furthermore, S_3, S_8, and S_{11} performed better with GPT-4o-mini, while S_5 yielded stronger results with Llama, suggesting that certain prompt combinations may be more suitable to specific LLM architectures.

Therefore, we may then answer our central research questions ("*Can the combination of prompt engineering techniques bring more effective results when using LLMs? Do some techniques that have not yet been much studied, have potential for more accurate results on such combination?*"). Our findings indicate that there is indeed potential in combining prompt techniques, rather than relying solely on the most commonly used prompts. We demonstrate that certain PE combinations, when applied to the Llama model, can outperform GPT-4o-mini and vice versa. Notably, through the formulated hypothesis test, we confirmed that our combined approach resulted in statistically significant improvements over the baseline.

7. Conclusion

This paper presented an evaluation of PE techniques, focusing on the potential of combining multiple techniques and assessing their effectiveness using the BERTScore metric. While the Role, RaR and RAG PE techniques demonstrated higher scores across two evaluation scenarios (in both GPT-4o-mini and Llama models), we do not claim that these combinations are categorically superior to the others. Instead, our findings reveal several alternative configurations that exhibit strong potential and may serve as valuable templates for future prompt engineering efforts. We emphasize the importance of exploring combinatorial approaches to PE, moving beyond the isolated use of individual techniques or the exclusive development of new frameworks. This direction may offer more versatile and robust solutions for LLM-based applications.

We observed that the Llama model offers more opportunities for experimentation compared to the OpenAI one, mainly due to its more accessible cost profile. It is worth mentioning that our study was limited by some constraints in terms of available computational resources and a relatively small set of evaluation questions, as scaling up the number of evaluation tasks would have significantly increased both the financial cost and processing time of API requests, making larger-scale experimentation impractical within the scope of this work.

This work establishes a foundation for future research in some directions: systematically testing additional combinations of PE techniques, developing standardized prompt templates to enhance LLM response accuracy, and extending evaluations to include other LLMs, such as Gemini and DeepSeek. Future work may also explore metrics like *Pass@K* and apply PE strategies to broader domains. Collectively, these directions have the potential to advance the field of prompt engineering and improve the practical capabilities of LLMs across various applications.

The code underlying this study has been made available in a GitHub repository[5].

[5]Combiner PE techniques

References

Bansal, P. (2024). Prompt engineering importance and applicability with generative ai. *Journal of Computer and Communications*, 12.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Coignion, T., Quinton, C., and Rouvoy, R. (2024). A performance study of llm-generated code on leetcode. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, pages 79–89.

Damke, G., Gregorini, D., and Copetti, L. (2024). Avaliação da performance e corretude na geração de código através de técnicas de engenharia de prompt: Um estudo comparativo. In *Anais do XXI Congresso Latino-Americano de Software Livre e Tecnologias Abertas*, pages 400–403, Porto Alegre, RS, Brasil. SBC.

Deng, Y., Zhang, W., Chen, Z., and Gu, Q. (2023). Rephrase and respond: Let large language models ask better questions for themselves. *arXiv preprint arXiv:2311.04205*.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dodge, Y. (2008). *The concise encyclopedia of statistics*. Springer Science & Business Media.

Gouveia, T., Albuquerque, K. M. M., Oliveira, J. D., and Maciel, V. M. B. C. (2023). C073: ferramenta para apoio ao ensino de programação usando a metodologia de aprendizagem baseada em problemas. *Revista Principia*, 60(1):70–87.

Hu, T. and Zhou, X.-H. (2024). Unveiling llm evaluation focused on metrics: Challenges and solutions. *arXiv preprint arXiv:2404.09135*.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Korzyński, P., Mazurek, G., Krzypkowska, P., and Kurasiński, A. (2023). Artificial intelligence prompt engineering as a new digital competence: Analysis of generative ai technologies such as chatgpt. *Entrepreneurial Business and Economics Review*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Mahir, A., Shohel, M. M. C., and Sall, W. (2024). *The Role of AI in Programming Education: An Exploration of the Effectiveness of Conversational Versus Structured Prompting*, pages 319–352. Practitioner Research in College-Based Education.

Medeiros, A., Cavalcante, C., Nepomuceno, J., Lago, L., Ruberg, N., and Lifschitz, S. (2024). Contrato360: uma aplicação de perguntas e respostas usando modelos de linguagem, documentos e bancos de dados. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 155–166, Porto Alegre, RS, Brasil. SBC.

Neves, B., Sousa, T., Coutinho, D., Garcia, A., and Pereira, J. (2024). Explorando o potencial e a viabilidade de llms open-source na análise de sentimentos. In *Anais Estendidos do XV Congresso Brasileiro de Software: Teoria e Prática*, pages 89–98, Porto Alegre, RS, Brasil. SBC.

Ning, X., Lin, Z., Zhou, Z., Wang, Z., Yang, H., and Wang, Y. (2024). Skeleton-of-thought: Large language models can do parallel decoding. *Proceedings ENLSP-III*.

Reynolds, L. and McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Sabit, E. (2023). Prompt engineering for chatgpt: a quick guide to techniques, tips, and best practices. *Techrxiv preprint 10.36227/techrxiv.22683919*.

Sarker, L., Downing, M., Desai, A., and Bultan, T. (2024). Syntactic robustness for llm-based code generation. *arXiv preprint arXiv:2404.01535*.

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., Dulepet, P. S., Vidyadhara, S., Ki, D., Agrawal, S., Pham, C., Kroiz, G., Li, F., Tao, H., Srivastava, A., Costa, H. D., Gupta, S., Rogers, M. L., Goncearenco, I., Sarli, G., Galynker, I., Peskoff, D., Carpuat, M., White, J., Anadkat, S., Hoyle, A., and Resnik, P. (2025). The prompt report: A systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608*.

Shin, J., Tang, C., Mohati, T., Nayebi, M., Wang, S., and Hemmati, H. (2023). Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks. *ArXiv*, abs/2310.10508.

Vatsal, S. and Dubey, H. (2024). A survey of prompt engineering methods in large language models for different nlp tasks. *ArXiv*, abs/2407.12994.

Wang, T., Zhou, N., and Chen, Z. (2024). Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation. *arXiv preprint arXiv:2407.05437*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Weng, Y., Zhu, M., Xia, F., Li, B., He, S., Liu, S., Sun, B., Liu, K., and Zhao, J. (2022). Large language models are better reasoners with self-verification. *arXiv preprint arXiv:2212.09561*.

Woolson, R. F. (2005). Wilcoxon signed-rank test. *Encyclopedia of biostatistics*, 8.

Zheng, K., Decugis, J., Gehring, J., Cohen, T., Negrevergne, B., and Synnaeve, G. (2024). What makes large language models reason in (multi-turn) code generation? *arXiv preprint arXiv:2410.08105*.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. (2022). Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.