How can large language models and prompt engineering be leveraged in Computer Science education?

Systematic literature review

Alexandra Ioana Neagu[1]

Supervisor(s): Fenia Aivaloglou[1]**, Xiaoling Zhang**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 24, 2023

Name of the student: Alexandra Ioana Neagu
Final project course: CSE3000 Research Project
Thesis committee: Fenia Aivaloglou, Xiaoling Zhang, Tom Viering

Contents

Abstract

In recent years, significant progress has been made in the field of natural language processing (NLP) through the development of large language models (LLMs) like BERT and ChatGPT. These models have showcased remarkable abilities across a range of NLP tasks. However, effectively harnessing their potential requires meticulous prompt engineering and a comprehensive understanding of their limitations.

Additionally, LLMs have attracted attention in the educational domain for their potential to enhance learning and teaching experiences, particularly in fostering the development of computational thinking skills.

This paper aims to explore the potential of leveraging NLP and prompt engineering techniques to generate successful solutions to coding problems following initial failures. Furthermore, the research explores the potential applications of NLP techniques in teaching and learning practices involving LLMs and their potential drawbacks in this context.

Keywords – Large Language Models (LLMs), Natural Language Processing (NLP), Prompt engineering, ChatGPT, Code generation, Education, BERT

1 Introduction

Large language models (LLMs) have made significant advancements in the field of natural language processing (NLP) in recent years [2]. They have demonstrated remarkable capabilities in various NLP tasks, including language generation, code generation, and automatic program repair. However, effectively utilizing LLMs requires careful prompt engineering and an understanding of their limitations.

LLMs, such as BERT and ChatGPT, which we will consider in this paper, differ in their underlying architectures and training methodologies. These architectural differences impact the capabilities and limitations of the models, making them more suitable for specific tasks [43].

Prompt engineering is the process of crafting input prompts to LLMs in order to produce desired responses. It involves strategies such as providing explicit instructions and leveraging implicit context. By carefully designing prompts, researchers and developers can enhance the accuracy, coherence, and contextuality of LLM-generated outputs [21].

On top of the NLP benefits that LLMs have brought forth, their application in the educational field has also garnered attention as an intriguing area with a multitude of possibilities. By leveraging these models, there exists the potential to enrich the learning and teaching experiences of individuals across various educational levels [16], including the development of one's computational thinking skills.

In this paper, we aim to investigate to what extent we can leverage NLP and prompt engineering techniques upon the input of an LLM in order to generate successful solutions to programming problems, after initial failure. Afterward, we consider how such strategies can be integrated into the educational environment in order to help students develop their computational thinking skills. We will treat the following research topics:

1. **RQ1**: What prompt engineering techniques are used to support problem solvers to modify the problem description successfully?
2. **RQ2**: What is the potential use of NLP techniques in teaching and learning practices that leverage LLMs?

Section 2 gives background knowledge required to better understand the research at hand. Section 3 provides an overview of the research's paper selection process, including the search criteria and methodology employed. Section 4 presents the findings of the aforementioned two RQs. Section 5 explores the implications of the findings and their significance within the field, and then delves into the ethical and responsible considerations that were taken into account

throughout the course of this research. Section 6 treats factors that pose a threat to the validity of this paper. Lastly, Section 7 presents a concise overview of the study's findings, followed by an exploration of potential opportunities for future research. A list of acronyms used frequently in this paper can also be found in Appendix A. Lastly, Appendix B provides an overview of the research sub-topics and themes discussed in this paper, including a concise summary of the pertinent references for each category, as presented in Table 1.

2 Background

In this section, we discuss the NLP techniques used in LLMs like BERT and ChatGPT. These LLMs are based on the Transformer architecture, which incorporates self-attention mechanisms. We discuss the encoder-only architecture of BERT and the decoder-only architecture of ChatGPT.

2.1 Transformers

LLMs are a class of deep learning models that appeared in recent years and majorly advanced the NLP field [43; 36]. Popular examples of such LLMs are OpenAI's ChatGPT, a generative pre-trained transformer (GPT) with the user interface of a chatbot, and Google's BERT (Bidirectional Encoder Representations from Transformers) [10]. As the names suggest, these LLMs are based on the Transformer architecture, a concept introduced in 2017 by *Vaswani et al.* [38]. The Transformer is a type of neural network, an innovative architecture that uses multiple layers of neural components to assign and distribute weight representations to each knowledge unit [38]. What sets it apart from previous neural network models, such as the recurrent neural network and the convolutional neural network, is that it is solely based on attention mechanisms and does not use recurrent or convolutional techniques. The attention mechanism is the component responsible for assigning weights to all the encoded input units and for learning which parts of the input have what importance in the context of the whole data. Therefore, this component plays the main role in the Transformer's ability to capture long-range dependencies in a sentence [10]. Furthermore, since Transformers do not employ recurrent techniques, this allows them to be parallelizable.

The inclusion of the attention mechanisms has proven to be advantageous in the NLP field, as this made Transformers outperform traditional convolutional and recurrent neural networks models on a variety of NLP tasks [31]. Many attention mechanisms have been developed and integrated into LLMs [11]. One such mechanism that has been observed to perform well on NLP tasks is the self-attention mechanism [46], which focuses attention on different parts of the input sequence and connects them in order to output a representation of the same sequence. Transformers implement this strategy by leveraging a mask matrix, which dictates which input tokens are visible between them. Subsequently, Transformers also make use of tokenization, an NLP technique that dictates how a given text input is divided into smaller units called tokens. Many tokenization methods exist, and Transformer models use a hybrid between word-level and character-level tokenization called subword tokenization [9]. These tokenization techniques go hand-in-hand with the encoder-decoder architecture, the most commonly used underlying layering in Transformers [1]. In this architecture, the encoder extracts features from the input sentence (using the attention mechanism), and the decoder uses the features to produce an output sentence (translation). Next, this subsection delves into the specific architectures of BERT and ChatGPT.

2.2 BERT

One common approach to training LLMs with extremely large, readily-available natural language datasets is the Masked Language Model (MLM). MLM is an unsupervised training paradigm where a model is fed a text input with masked words in it, and it must predict these words based on the surrounding context. BERT is pre-trained using MLM, which allows it to gain a deeper understanding of the relationships between tokens and the overall context they form [43]. As for the architecture BERT is built upon, it uses a stacked multi-layer bi-directional Transformer, and WordPiece [41] as a word segmentation method (tokenization technique) [6]. The bi-directional property of the model refers to the fact that an evaluated token has access to both its left and right contexts in the sentence, providing a bi-directional representation. This in turn gives BERT a better context extractor for reasoning tasks, such as solving mathematical problems or writing cde [10]. However, the input is encoded bi-directionally, and masked tokens are predicted independently from each other, which reduces the generation ability [19]. BERT is also an encoder-only model. Due to its pre-training that focuses on downstream tasks [31], the (bi-directional) encoder is trained to generate a fixed-length representation of the input text, and this representation is used as input to a downstream task that is responsible for generating text as output. The BERT model itself does not have a decoder component and is not designed to generate text directly.

2.3 ChatGPT

ChatGPT, which is built on top of OpenAI's GPT-3.5 and GPT-4 foundational GPT LLMs, uses an autoregressive stacked left-to-right Transformer for its decoder as a feature extractor [31; 30]. An autoregressive language model (LM) predicts the next word in a sequence of words based on the words that have come before it. It is a uni-directional pipeline, it can only reach the left context of the evaluated token, hence the name left-to-right [10]. So ChatGPT is decoder-only since GPT2 updated its architecture to decoder-only, as the decoder became equivalent to the encoder. This property, combined with fine-tuning the model using downstream tasks, makes this architecture more appropriate for text-generation tasks [46].

3 Methodology

This section describes the search criteria and methodology process [35; 17] used to select the papers for this systematic literature review.

3.1 Search criteria & filtering

The *Google Scholar* engine was used to search papers focused on the previously introduced two sub-topics. Only studies in English were considered, as this is the language that the author of this work is most familiar with and it is also the language that most of the available papers were written in. The following search queries and corresponding filters were used:

- (”natural language processing” OR NLP) AND (”large language models” OR LLM) AND ”program synthesis”
  - 276 results were returned
- (”large language models” OR ”large language model” OR LLM) AND ”program synthesis” AND (”prompt” OR ”prompts” OR ”prompt optimisation”)
  - 144 results were returned
- (”large language models” OR ”large language model” OR LLM) AND (”education” OR ”teaching” OR ”learning”) AND ”program synthesis”
  - 354 results were returned

In Fig. 1 we present the PRISMA flow chart [28] of the record selection process done.

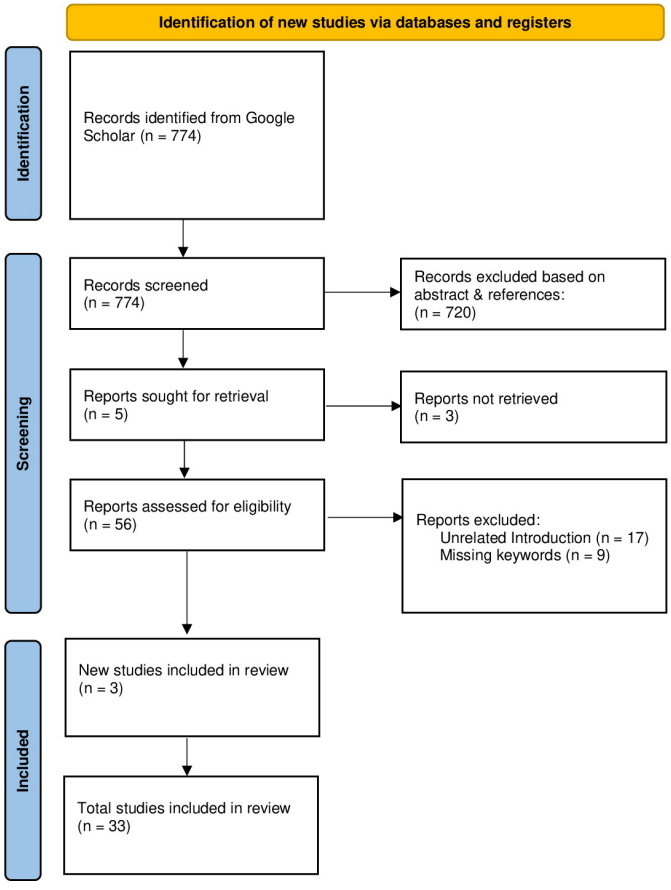

Figure 1: Flow chart of included studies. The numbers exclude duplicated papers.

All the queries were conducted on 2023.05.11 and had the ”include citations” checkbox ticked and the filtering set to ”Since 2023”. The papers that were examined to determine if they were suitable or not were only from the first three pages of the returned results.

3.2 Inclusion & exclusion criteria

The following *inclusion criteria* were established for the selection of studies in this systematic literature review:

- Topic Relevance: Studies investigating either of the two sub-questions, or both. Keywords were used to determine the relevance of the paper, as per the coding and labeling procedure.
- Language: Studies published in English for comprehension and accessibility.
- Publication Date: Studies published from 2020 to the present to focus on recent research.

If a paper did not abide by all of these inclusion criteria, it was excluded.

After this process, the papers that were gathered were supplemented with other papers found through different means. Other relevant papers were found using the backward snowballing technique [13], by analyzing the *References* section of the papers deemed relevant from the search queries and choosing them based on the aforementioned process. Lastly, three papers were also recommended by the author’s responsible professor and supervisor.

3.3 Analysis

To select papers relevant to the topics, the titles, abstracts, and references of the studies were read. If these did not contain enough information to accurately determine if they were suitable or not, the *Introduction* section was also read, and the paper was scanned in its entirety to see if the needed keywords were treated in the paper in appropriate contexts. Afterward, a rigorous analysis process was employed to examine the selected papers:

1. **Data Extraction**: Relevant information from each included study was systematically extracted, including study characteristics (e.g., authors, publication year, study design, topic) and key findings.
2. **Quality Assessment**: The quality and validity of the included studies were evaluated using established criteria specific to the study design (e.g., sample size, sampling method, data collection methods, data analysis, reporting and transparency). This ensured that only high-quality studies were considered in the analysis.
3. **Synthesis of Findings:** The extracted data and key findings were synthesized using a thematic approach. The first sub-question, and namely the most comprehensive one, was divided into 4 themes: Prompt engineering, Constraints, Prompting techniques, and Heuristic strategies. Similarities, patterns, and discrepancies among the studies were identified to gain insights into the research sub-questions.

4. **Interpretation:** The synthesized findings were interpreted in the context of the research sub-questions. The strengths, limitations, and implications of the collective evidence were considered to draw meaningful conclusions.

4 Results

In this section, we present the results of the research on the two sub-questions.

4.1 RQ1: The prompt engineering techniques used to support problem solvers to modify the problem description successfully

Even the most advanced models, fine-tuned for specific programming tasks, necessitate an expensive filtering process. This step involves discarding outputs from LLMs that fail to compile or pass tests [20]. These discarded outputs often exhibit surface-level similarities to correct solutions [32], despite failing to produce the expected output. This challenge is commonly referred to as the "near miss syndrome" or the "last mile problem" [3; 22].

In this subsection, we explore prompts for problem-solving and the importance of prompt engineering. We discuss heuristic strategies, the interpretability of prompts as constraints on output generation, and recent insights on prompt interpretation and difficulty.

4.1.1 Prompt engineering

LLMs have the ability to generate code based on natural language descriptions [12]. Transforming such natural language specifications to code is considered a form of inductive specification [45]. LLMs, models that take such natural language specifications as input, have demonstrated remarkable success in the field of automatic program repair [7]. To effectively harness the full potential of these LLMs, prompts, the input text received by the model, need to be carefully crafted in order to produce desired responses. Therefore, the rise in popularity of these models has elicited the concept of prompt engineering. This process is not only beneficial for generating accurate, coherent, and contextually appropriate responses but also for understanding the capabilities and limitations of the model in question [5]. Furthermore, this helps improve the explainability of the models, a characteristic highly valued in the field of Explainable Artificial Intelligence.

The process of prompt engineering entails various strategies, including explicit instruction, and implicit context [21]. Explicit instruction involves providing explicit guidance or constraints to the model through instructions, examples, or specifications. Implicit context leverages the model's understanding of the preceding context to influence its response, which, as we have seen in the previous section, LLMs excel at because of their underlying architecture [23].

The effectiveness of prompt engineering lies in striking a balance between providing sufficient guidance to achieve the desired output while allowing the model's training and language generation capabilities to flourish. It requires careful consideration of factors such as the length and complexity of prompts, the type and structure of questions asked, and the intended context or domain of the conversation [15].

4.1.2 Constraints

One way to categorize prompts is based on the constraints they define. For example, the prompt "Write a function in Java that sorts the provided array and a short simple explanation of how it works" contains a multitude of constraints. The generated output must be a "function in Java" (*code-style constraint*), alongside an explanation (*document-type constraint*). The explanation must be "short" (*structural constraint*), and formulated in "simple" terms (*stylistic constraint*), and both the code and the explanation need to be about sorting the array that was given to the model (*subject constraint*). Therefore, this partitioning of constraints can be defined, as all prompts can be viewed as combinations of different constraint types [24].

Recent studies, such as *Lu et al., 2023* [24], observed that especially stylistic and structural constraints have a high influence on the quality of the output. Stylistic constraints refer to guidelines regarding the tone, formality, or specific writing style to be followed. Structural constraints, on the other hand, govern the organization, coherence, and overall structure of the output. These constraints are widely encountered across prompts and prove challenging to engineer. While incorporating them individually might seem manageable, the interaction between different stylistic and structural constraints can lead to unexpected and diverse outputs, making it difficult to model their combined influence accurately [27].

In their study, *Lu et al., 2023* [24] also discovered some interesting results about GPT-3's prompt interpretation:

- GPT-3 relies on incidental connections between style and subject matter, leading to spurious correlations. Some style constraints may be incompatible with writing certain types of content due to negative correlations.
- GPT-3 sometimes confuses style with subject when prompted with complex inputs. This is more common with less used styles, and it tends to treat the style itself as the subject in uncertain situations.
- GPT-3 encounters difficulties with words that are not exclusive to creative writing, due to an imbalance in the training dataset between creative and functional text.
- The difficulty perceived by humans in the prompt does not correlate with GPT-3's performance, indicating differing factors influencing difficulty between humans and LMs.
- GPT-3 faces challenges with numerical constraints, often failing to generate text within the required length but approximating the desired numerical value.
- Descriptive structural limitations, such as "long" or "short," result in varied outputs with overlapping lengths, likely influenced by the relative and context-dependent nature of length definitions.

4.1.3 Prompting techniques

Outside of seeing prompts as a string of tokens, where each individual token, or group of tokens, is interpreted as a specific category of constraints, prompts can also be viewed as tools for advanced prompting engineering techniques that allow us to achieve more complex and interesting tasks. Modern LLMs, like GPT-3, are trained to adhere to instructions

and have been extensively trained on vast datasets. As a result, these models possess the ability to perform certain tasks without any explicit training on those specific tasks, known as ”zero-shot” capability [39]. On the other hand, there is ”few-shot” prompting, that can serve as a method to facilitate in-context learning, allowing us to enhance the model’s performance by providing demonstrations within the prompt. These demonstrations act as conditioning for subsequent examples, guiding the model in generating desired responses [8]. LLMs have demonstrated remarkable proficiency in ”few-shot” learning, displaying the ability to learn from a limited number of examples [5]. However, they tend to be less effective in ”zero-shot” learning scenarios. GPT-3, for instance, exhibits significantly lower performance in ”zero-shot” tasks such as context comprehension, problem-solving, and natural language processing compared to ”few-shot” tasks. One possible explanation for this discrepancy is that without a ”few-shot” training paradigm, models struggle to excel in prompts that deviate from the format of the pre-training data. On top of this, it has also been observed that stylistic constraints prove hard to engineer in ”zero-shot” prompting contexts [33].

Recent research has highlighted a substantial disparity between the universal knowledge embedded in LLMs and the specific behavioral patterns exhibited by users within private domain data [44]. Hence, ”zero-shot” and ”few-shot” engineering methods may be insufficient to deploy LLMs in specific private task systems. In the paper ”*Fine-tuned Language Models Are Zero-Shot Learners*” (FLAN) by Google [39], a simple approach to enhance the ”zero-shot” performance of LLMs is introduced, thus extending their applicability to a wider user base. Instruction tuning is a straightforward technique that combines advantageous elements from both the pretrain–finetune and prompting approaches. By employing supervision through finetuning, it aims to enhance the LM’s responsiveness to text interactions during inference. In Fig. 2 we can see a diagram comparison between the 3 popular approaches to LM tuning.

Pretrain-finetuning generally requires many task-specific examples in order to properly perform the desired task, and the model ends up specializing in that task and performing poorly in other instances (e.g. BERT). In the case of the GPT-3 engine, prompt engineering is employed to improve the performance of its generative capabilities via ”few-shot” training. At last, instruction tuning combines these two previous approaches by training the model to perform many different tasks via natural language instructions, and inferring from that how to solve new, unseen tasks.

Another strategy that has shown positive results is *Chain-of-thought* (CoT) prompting, a technique proposed by Google researchers, which improves the reasoning ability of LLMs by allowing them to derive the final answer to a multi-step problem by prompting them to generate a sequence of intermediate steps [40].

4.1.4 Heuristic strategies

In their study, *Jiang et al.* [15] tested how participants cope with model failures when generating code and the repair strategies they resort to. The ”few-shot” prompts they use

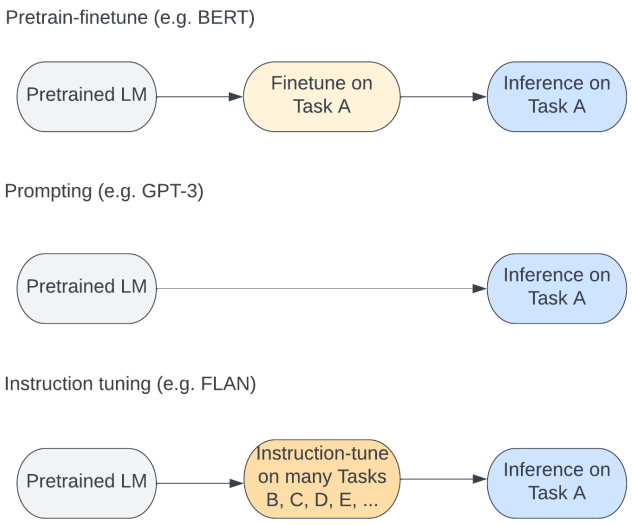

Figure 2: Comparing the process of pretrain–finetuning, prompting, and instruction tuning.

combine natural language and code in their input. They refer to this type of input as mixed inputs, which signifies that users can blend different types of input, such as code and natural language, in their requests to the model. Interestingly, it was discovered that the prompts do not necessarily need to include examples that mix natural language and code in the description field. Even without such examples, LLMs can often accurately interpret mixed inputs and generate code as output. Upon initial failure, the heuristic approaches that were mainly used, both by novice and expert coders, were: reword (add, drop, change, or reorder words), expand the scope of the request, retry (rerun the same request unchanged), reduce the scope of the request, re-calibrate specific targets with ”easier” targets. In general, they did not find any specific strategy that consistently increased the likelihood of participants accepting the generated code as correct.

However, a strategy that has proven beneficial for code output repair is a conversational approach [42; 15]. This form of interaction, where the model assists the user in deriving specifications in a reply exchange manner, aligns naturally with user expectations, and it can support novices in breaking down complex problems into more manageable components.

In their paper, *Liventsev et al.* [22], found out that GTP-3 achieves the highest performance for code completion in C++ instructions when the input contains the word ”obviously”. In general, they observed that GPT-3 performs poorly in C++ when the input instruction includes verbs such as ”yield” and ”ensure,” which are likely infrequently used in code documentation.

4.2 RQ2: The potential use of NLP techniques in teaching and learning practices that leverage LLMs

In order to explore the viability of pair programming with an intelligent agent, *Kuttal et al.* [18] conducted a Wizard-of-

Oz study. The findings suggest that agents can be valuable partners in pair programming, helping overcome expertise-related challenges, although there may be a trade-off in terms of code creativity.

In another study, *Brandt et al.* [4] demonstrated the integration of web search into a development environment to simplify the process of discovering and applying web-based examples to code. By utilizing a generative model trained on a code-inclusive corpus in a similar way, comparable outcomes can be achieved for certain information-seeking requirements. Specifically, users can express their intentions using natural language, and the generative language model can produce the relevant code. This concept is exemplified by GPT-3 demonstrations that convert natural language into code or continue writing code on behalf of the user. In this scenario, the natural language input to the model functions akin to a search query, but instead of returning web pages, the model generates code.

The idea of leveraging NLP techniques to influence the generation of better code outputs was also explored in the study of *Jiang et al.* [15]. Due to the unexpected responses of each model, participants in the study felt that in order to generate the desired output, one needs to familiarize themselves with the model's syntax, including specific words and phrases it is more likely to interpret accurately. Despite this, participants expressed optimism regarding the potential of using natural language to overcome barriers in interacting with technology, and that the user group who would benefit most from such interactions with code-generation models could be individuals who have some coding knowledge but lack expertise in a specific programming language, struggle with syntax, or have limited experience with a particular library or API.

GPT-3 exhibits greater reliability in detecting erroneous code compared to its ability to generate from scratch code without errors [33], which hints that providing it snippets of related code in the input may increase its efficiency in producing a correct code output.

In the field of computer science, *MacNeil et al.* [25] have utilized GPT-3 to generate code explanations. While there are still open research and pedagogical questions that require further investigation, this work has effectively showcased the potential of GPT-3 in aiding learning by providing explanations for various aspects of a given code snippet.

Another LLM that has been tested in education settings is OpenAI's Codex, a descendant of the GPT-3 model, fine-tuned for use in programming applications. Through "few-shot" learning, *Sarsa et al.* [34] demonstrated that the OpenAI Codex model can offer a range of programming tasks along with their corresponding correct solutions, automated tests to validate students' solutions, and additional code explanations. In general, fine-tuning LLMs on domain-specific corpora, such as IT documents and code, enables them to generate domain-specific language and better support learners in this domain [16].

In the study of *Megahed et al.* [26], ChatGPT also proved successful in solving programming tasks and facilitating the learning of new programming languages. However, a contributing factor to its success in this context can be attributed to the use of widely recognized packages with extensive online documentation in the input prompt.

5 Discussion

In this section, we first discuss the implications of the findings and their significance for the field, and then the ethical and responsible considerations that were taken into account when conducting this research.

5.1 The findings and their implications

The background information first reveals the differences in architecture between BERT and ChatGPT. ChatGPT's autoregressive nature makes it more suitable for text generation tasks, while BERT's encoder-only design makes it effective for downstream tasks that involve generating fixed-length representations of input text. This proves to show that certain models perform better on certain tasks, and when choosing an LLM to integrate into an educational environment, its strengths and weaknesses need to be taken into account.

The findings explored prompt engineering techniques used to support problem solvers in producing accurate and contextually appropriate responses from LLMs. The length and complexity of prompts, the type and structure of questions asked, the intended context or domain of the conversation, and the confidence of the tone are all indicators that need careful consideration. Constraints also play a significant role in prompt engineering, and recent studies have shed light on the influence of different constraint types. Stylistic and structural constraints were found to have a high impact on the quality of output, and their combined influence can lead to unexpected and diverse outputs. LLMs, such as GPT-3, exhibit sensitivity to style-subject constraint pairings and struggle with descriptive structural limitations. Understanding these constraints and their interactions can help in designing more effective prompts.

Moreover, the review highlighted the distinction between "zero-shot" and "few-shot" prompting. While LLMs possess the ability to perform certain tasks without explicit training ("zero-shot" capability), they tend to be less effective in such scenarios. "Few-shot" prompting, which provides demonstrations as conditioning for subsequent examples, has been shown to enhance model performance. For generating more accurate, step-by-step explanations of code, one can leverage "few-shot" prompting in combination with CoT.

The findings of this study also shed light on the potential use of NLP techniques in teaching and learning practices that leverage LLMs. These findings suggest that LLMs have the ability to assist in pair programming, code generation, code explanations, and programming language learning, offering promising opportunities for educational applications. However, it is important to note that there may be a trade-off in terms of code creativity when relying on LLMs in such contexts.

The study also found that familiarity with the LLM's syntax, including specific words and phrases it is more likely to interpret accurately, is crucial for achieving the desired output and for correcting upon initial failure in the response. LLMs have the potential to support individuals with coding knowledge but lack expertise in specific programming languages

(such as R [26]), struggle with syntax, or have limited experience with particular libraries or APIs. It is also worth noting that the use of widely recognized packages with extensive online documentation in the input prompt can contribute to the quality of the output. This highlights the importance of providing LLMs with relevant and comprehensive information to enhance their performance in programming-related tasks.

The findings also highlight the reliability of LLMs in detecting erroneous code compared to their ability to generate code from scratch without errors. This suggests that providing LLMs with snippets of related code in the input may enhance their efficiency in producing correct code outputs. This insight can inform the design of prompts or inputs provided to LLMs to improve their performance in generating accurate and error-free code.

The ease of generating information through LLMs can potentially have a detrimental effect on critical thinking and problem-solving skills. This is because the model simplifies the process of acquiring answers or information, which can foster laziness and discourage learners from conducting their own investigations and arriving at their own conclusions or solutions. To mitigate this risk, it is crucial to acknowledge the limitations of LLMs and utilize them as tools to support and enhance learning, rather than relying on them as substitutes for human authorities and other authoritative sources [29]. Such limitations include but are not limited to lack of interpretability, the potential for bias, unexpected brittleness even in relatively simple tasks, and hallucinations, a known limitation of current generative AI models where they output a completely made-up answer [14]. In terms of biases, at the heart of these LLMs lies the training process, which involves billions of lines of code extracted from open-source projects, including public GitHub repositories. Although these sources significantly contribute to the performance of LLMs, they often harbor security vulnerabilities resulting from insecure API calls, outdated algorithms/packages, insufficient validation, and subpar coding practices, among other issues [37].

5.2 Responsible research

In conducting this systematic literature review, ethical considerations were taken into account throughout the research process. The research question was carefully formulated to ensure that the study would not cause harm or discrimination to any individuals or groups. The study solely focused on the impact of natural language modifications to problem descriptions on the generation of successful code solutions, and no data related to personal information or sensitive topics were collected.

Moreover, the search criteria and methodology process used for the selection of papers were designed to be transparent and reproducible. By using a well-known and widely available search engine such as *Google Scholar* and specifying the search terms and filters, the process can be easily replicated by other researchers. The criteria for inclusion and exclusion of papers were carefully defined and reported in detail in the methodology section, making it possible for readers to assess the validity and accuracy of the selection process.

Furthermore, the process of selecting papers was carried out with a high level of rigor and transparency. To ensure that only relevant and appropriate papers were included in the review, the titles and abstracts of all the results were read and analyzed, and the full texts of papers were scanned when necessary. Additionally, the use of the backward snowballing technique and recommendations from the author's supervisors helped to ensure that a comprehensive and unbiased collection of papers was used.

In summary, this systematic literature review was conducted in a responsible and ethical manner, and the methodology used was designed to be transparent and reproducible. By adhering to these principles, the research can contribute to the advancement of knowledge in the field, while also maintaining ethical standards and upholding the integrity of the research process.

6 Threats to validity

While the author of this paper made proper efforts to ensure the validity of this research, there are still potential threats to validity that need to be taken into consideration.

Firstly, the study was conducted within a time frame of 8 weeks, a quite limited amount of time to properly delve into such a complex topic. Due to this time limitation, newer studies that have been published since the end of the search period may have been missed. This is a particularly important threat to consider, as this subject is a relatively new area of research, so one can expect more extensive research to appear in the future.

Secondly, building upon the previous threat, the study relied on a limited number of sources, and it is possible that important research on the topic may have been missed. Despite the inclusion criteria being carefully defined and the author being as methodical as possible in selecting the sources most relevant to the research topic, it is still possible that some relevant studies were excluded.

Additionally, the author considered only literature written in English, as this is the language they were most comfortable with and that yielded the most papers found. However, this leaves space for a certain degree of language bias, leading to an incomplete or biased view of the available literature.

Another significant limitation is the absence of an analysis of Google's Bard, a highly anticipated LLM. Due to its recent launch, there was limited availability of research and comprehensive evaluation, making it difficult to incorporate Bard's findings within the scope of this paper.

Relying solely on Google Scholar for paper retrieval may also limit the inclusiveness of the search. Other databases or platforms might have contained relevant publications that were not captured. Likewise, considering only the first three search result pages may introduce a bias by potentially excluding relevant papers beyond those pages.

Lastly, it is worth noting that the author of this study is not an expert in NLP techniques, code generation models, or LLMs. Consequently, there may be limitations on the choice of papers included in this study, and to the depth to which certain sub-topics were treated.

Despite these limitations, it is hoped that this study will contribute to a better understanding of the use of NLP techniques in code generation and stimulate further research in

this area.

7 Conclusions & future work

In this last section, we provide a summary of the findings of this study, followed by a discussion of potential avenues for further research.

7.1 Conclusions

The findings highlight the advancements in NLP techniques brought about by LLMs and the potential for integration into educational environments.

Overall, the findings suggest that NLP techniques in conjunction with LLMs have significant potential in enhancing teaching and learning practices in the field of computer science. A number of prompting techniques and their impact on the quality of the response have been investigated. Factors such as the prompt's length, complexity, question types, and constraints, play a significant role in obtaining accurate responses. Notably, "few-shot" prompting shows promise in improving LLM performance, especially in combination with CoT. By leveraging these techniques, LLMs can serve as valuable partners in pair programming, aid in code generation and explanations, support programming language learning, and facilitate the completion of programming tasks. However, it is important to consider factors such as code creativity trade-offs, familiarity with the LLM's syntax, and the provision of relevant information to optimize the use of LLMs in educational settings.

7.2 Future work

Future work should focus on refining prompt engineering techniques, exploring the potential of "few-shot" prompting in various educational scenarios, and investigating ways to strike a balance between LLM assistance and fostering independent thinking. Research is also needed to address the limitations of LLMs, including interpretability, bias mitigation, and improving their performance in complex programming tasks. Furthermore, efforts should be made to ensure the security and reliability of LLMs by addressing vulnerabilities and coding practices in the training process. By addressing these challenges and capitalizing on the potential of NLP techniques and LLMs, the integration of these technologies into educational settings can significantly enhance teaching and learning experiences.

A Acronyms

In this section, we present a list of the acronyms that are used frequently throughout this paper.

BERT = Bidirectional Encoder Representations from Transformers
GPT = Generative Pre-trained Transformer
LLM = Large Language Model
LM = Language Model
MLM = Masked Language Model
NLP = Natural Language Processing
RQ = Research Question

B Surveyed research

In this section, we present in Table 1 an overview of the research sub-topics and themes addressed in this paper, with a summarized list of the relevant references for each category.

Table 1: Surveyed Research on the RQs

RQ	Themes	References
Background	Transformers	[1], [2], [9], [10], [11], [31], [36], [38], [43], [46]
	BERT	[6], [10], [19], [31], [41], [43]
	ChatGPT	[10], [30], [31], [46]
RQ1	Prompt engineering	[3], [5], [7], [12], [15], [20], [21], [22], [23], [32], [45]
	Constraints	[24], [27]
	Prompting techniques	[5], [8], [33], [39], [40], [44]
	Heuristic strategies	[15], [22], [42]
RQ2	-	[4], [14], [15], [16], [18], [25], [26], [29], [33], [34], [37]

C References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[3] Rohan Bavishi, Harshit Joshi, José Cambronero, Anna Fariha, Sumit Gulwani, Vu Le, Ivan Radiček, and Ashish Tiwari. Neurosymbolic repair for low-code formula languages. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):1093–1122, 2022.

[4] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R Klemmer. Example-centric programming: integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 513–522, 2010.

[5] T Brown, B Mann, N Ryder, M Subbiah, JD Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, et al. Language models are few-shot learners in advances in neural information processing systems (eds larochelle, h., ranzato, m., hadsell, r., balcan, mf & lin, h.) 33 (curran associates, inc., 2020), 1877–1901. *arXiv preprint arXiv:2005.14165*, 2020.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Zhiyu Fan, Xiang Gao, Abhik Roychoudhury, and Shin Hwei Tan. Automated repair of programs from large language models. *arXiv preprint arXiv:2205.10583*, 2022.

[8] Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

[9] Henry Gilbert, Michael Sandborn, Douglas C Schmidt, Jesse Spencer-Smith, and Jules White. Semantic compression with large language models. *arXiv preprint arXiv:2304.12512*, 2023.

[10] Anthony Gillioz, Jacky Casas, Elena Mugellini, and Omar Abou Khaled. Overview of the transformer-based models for nlp tasks. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 179–183. IEEE, 2020.

[11] Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R Martin, Ming-Ming Cheng, and Shi-Min Hu. Attention mechanisms in computer vision: A survey. *Computational Visual Media*, 8(3):331–368, 2022.

[12] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

[13] Samireh Jalali and Claes Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 29–38, 2012.

[14] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[15] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J Cai, and Michael Terry. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2022.

[16] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023.

[17] Khalid S Khan, Regina Kunz, Jos Kleijnen, and Gerd Antes. Five steps to conducting a systematic review. *Journal of the royal society of medicine*, 96(3):118–121, 2003.

[18] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. Trade-offs for substituting a human with an agent in a pair programming context: the good, the bad, and the ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2021.

[19] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[20] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[21] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[22] Vadim Liventsev, Anastasiia Grishina, Aki Härmä, and Leon Moonen. Fully autonomous programming with large language models. *arXiv preprint arXiv:2304.10423*, 2023.

[23] Robert L Logan IV, Ivana Balažević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint arXiv:2106.13353*, 2021.

[24] Albert Lu, Hongxin Zhang, Yanzhe Zhang, Xuezhi Wang, and Diyi Yang. Bounding the capabilities of large language models in open text generation with prompt constraints. *arXiv preprint arXiv:2302.09185*, 2023.

[25] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2*, pages 37–39, 2022.

[26] Fadel M Megahed, Ying-Ju Chen, Joshua A Ferris, Sven Knoth, and L Allison Jones-Farmer. How generative ai models such as chatgpt can be (mis) used in spc practice, education, and research? an exploratory study. *arXiv preprint arXiv:2302.10916*, 2023.

[27] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[28] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. The prisma 2020 statement: an updated guideline for reporting systematic reviews. *International journal of surgery*, 88:105906, 2021.

[29] John V Pavlik. Collaborating with chatgpt: Considering the implications of generative artificial intelligence for journalism and media education. *Journalism & Mass Communication Educator*, page 10776958221149577, 2023.

[30] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[31] Abir Rahali and Moulay A Akhloufi. End-to-end transformer-based models in textual-based nlp. *AI*, 4(1):54–110, 2023.

[32] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

[33] Laria Reynolds and Kyle McDonell. 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, 2021.

[34] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43, 2022.

[35] Hannah Snyder. Literature review as a research methodology: An overview and guidelines. *Journal of business research*, 104:333–339, 2019.

[36] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. Is chatgpt the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938*, 2023.

[37] Catherine Tony, Markus Mutas, Nicolás E Díaz Ferreyra, and Riccardo Scandariato. Llmseceval: A dataset of natural language prompts for security evaluations. *arXiv preprint arXiv:2303.09384*, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[41] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[42] Chunqiu Steven Xia and Lingming Zhang. Conversational automated program repair. *arXiv preprint arXiv:2301.13246*, 2023.

[43] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712*, 2023.

[44] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. Recommendation as instruction following: A large language model empowered recommendation approach. *arXiv preprint arXiv:2305.07001*, 2023.

[45] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 627–648, 2020.

[46] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.