Policy Research Journal
ISSN (E): 3006-7030 ISSN (P) : 3006-7022 Volume 3, Issue 9, 2025

ANALYZING THE CONSEQUENCES OF PROMPTS ON SOFTWARE DEVELOPERS USING DEEP LEARNING AND GEN-AI

Muhammad Murtaza Khan[1], Naeem Aslam[2], Umair Rashid[*3], Ahmad Naeem[4], Syed Shahid Abbas[5], Mohsin Ikhlaq Jaam[6]

[1,2,4,5,6]*Department of Computer Science, NFC Institute of Engineering and Technology, Multan, Pakistan*
[*3]*Department of Computer Science, Bahauddin Zakeriya University Multan, Pakistan*

[1]tareenk168@gmail.com, [2]naeem.aslam@nfciet.edu.pk, [*3]umairrashiid@gmail.com, [4]ahmad.naeem@nfciet.edu.pk, [5]syedshahid.abbas100@gmail.com, [6]mohsinikhlaq37@gmail.com

DOI: https://doi.org/10.5281/zenodo.17090135

Keywords
Generative AI, prompt engineering, software development, AI-generated code, developer satisfaction

Article History
Received: 18 June 2025
Accepted: 28 August 2025
Published: 10 September 2025

Corresponding Author: *
Umair Rashid

Abstract

This paper reports on an empirical investigation of the effects of early engineering on performance of Generative artificial intelligence (Gen-AI) tools for software development. Specifically, this thesis explores how ambiguous, purpose-specific, and incremental prompts can be used to improve AI-generated code to be more truthful, productive, and developer satisfying. Our experiments find that the structured prompts substantially outperform the vague prompts in different metrics. Goal-specific prompts approached a 25% improvement in accuracy while cutting down task completion in half, and step-by-step prompts sequenced a 40% improvement in accuracy while cutting down task completion in half. Using structured prompts led to an increase in developer satisfaction of between 30 - 40%, implying a desire for improved and more direct instructions. In addition to accuracy and time efficiency, structured prompts provided a 20% to 25% gain in precision and recall measures, with step-by-step prompts providing the greatest improvements in both measures. For step-by-step prompts, F1-scores were always larger, meaning a more balanced performance between recall and precision. The use of goal and step-by-step prompts was also associated with a 25-30% reduction in error rates, further validating goal and step-by-step prompts as a way to reduce the need for correction. However, this paper presents, for the first time, a systematic comparison of various types of prompts and the quantitative analysis of their contribution in piece writing in AI-generated code. By empirically demonstrating the effects of prompt structure on the performance of Gen-AI agents, this paper contributes concrete evidence for how to effectively structure human-AI collaboration in software development. Results show that early engineering is a critical enabler in realizing the true potential of Gen-AI tools and improving developer productivity and satisfaction.

INTRODUCTION

Gen-AI (Generative AI) tools have been introduced as a disruptive technology in software development community and have altered the manner in which developers undertake the task of writing code

radically. Deep learning is being applied to aid software developers in writing code, refactoring and debugging with projects like GitHub Copilot, OpenAI Codex project, etc. all which are powered by transformer-based architectures. Since the use of these tools has become commonplace, another challenge and opportunity has arisen, and the manner in which the prompts are formulated is one of the most decisive parameters that precondition the effectiveness of Gen-AI tools. A prompt is simply the input that an AI system receives and would guide it to produce the desirable output and the quality of the prompt is a direct reflection on the output AI produced code [1]. Although it is evident that Gen-AI tools have a bright future in the sphere of software development, the interaction of developers with them via prompt engineering is still mentioned. It is this gap that this work attempts to address by analyzing the effects of various types of prompts, both short and unstructured and much more structured and iteration-based on the generation of AI code. The method of optimizing AI-assisted software development to maximize the desirable code correctness, efficiency, and developer satisfaction can be learned through the understanding of how timely formulation is related to these results.

In the current literature, attention is paid to the abilities of Gen-AI tools to produce code, replicating repetitive tasks, and improving the developer's productivity. For example, as reported by [2] and [3] ,Gen-AI technologies like OpenAI's Codex and GitHub Copilot can greatly speed up coding, reduce programming errors, and offer programmers helpful code suggestions. They can be used to generate snippets of code, fix bugs and even suggest optimizations while you're working in an editor, and so are invaluable when working in a modern software engineering setting. However, to date, the majority of research on code search has concentrated on the tools themselves, such as the technical abilities of the models used (model accuracy, training on large repositories of code, etc.). What has largely been unknown is how timely formulation can affect the generation of output from these kinds of models. In particular, most studies have used fixed or predetermined prompts, without considering how the different prompt forms - vague, specific or step-by-step - impact on the quality of the generated code. This lacuna in the literature indicates an area for prompt engineering research, which, despite the substantial promise towards achieving better performance and developer productivity, has so far attracted little focus[4].

Prompt Engineering Prompt engineering or the art of building and optimizing prompts to get the best and most reliable answers out of AI models has emerged as a prerequisite skill among the developers of Gen-AI tools. In the software engineering context, timely engineering is at the centre of the good or bad quality and reliability of the AI-generated code. And, as the work of contemporary software development, including debugging, writing algorithms and connecting API increases significantly in complexity, developers are working with AIs in increasingly complicated manners. A bad prompt may lead to the creation of vaguity and incompleteness or incorrect code that then leads to more debug time and the possibility of introducing bugs into the code-base. Conversely, through well considered prompts the AI model can be assisted to jump start its thinking process of constructing highly relevant and functional code; which makes the entire development process more efficient [5]. The difficulty with the developers is not just to write correctly functioning code, but to formulate their intentions clearly and in a way narrow enough to be comprehended by the AI system. This exploration of prompt engineering in software engineering is interesting since it can result in more efficient human-AI interaction and an increase in the usefulness of Gen-AI tools in real-world code development projects.

A modified data set imitating real world software development problems will be used in this work in an attempt to better comprehend the impact of early formulating on AI generated code. The corpus will contain several coding tasks of the type typically present in the software engineering environment, which include: implementation of an algorithm, debugging and integrating API. The three categories of prompts that will be used to introduce each of the coding tasks will be vague prompts that do not provide much in the way of context; goal-oriented prompts that have specific directions; and step by step prompts that go through a sequence of smaller, more manageable tasks step-by-step. Such differences

will assist the study to determine the discrepancies in performance between various types of prompts in terms of accuracy, efficiency, and quality of codes generated by the AI [6]. This study will allow significant contribution to the understanding of what kinds of prompts are the most successful in Gen-AI-assisted software development by exposing developers to different types of prompts in a systematic fashion and quantitatively and qualitatively measuring their effectiveness.

The methodology herein described will be translated into experiment through posing professional developers and advanced students in computer science to perform software-development tasks with the aid of Gen-AI technologies like GitHub Copilot and ChatGPT. Respondents will be selected at random to either one of three types of prompts and their performance will be evaluated based on particular standardized indicators (code accuracy, time to complete task, and satisfaction with overall experience as a developer). Besides, the study will adopt the use of both automated tools and human reviewers in the process of determining the quality of the code generated by AI. In this manner, both technical (e.g. correctness, precision, recall) and human (e.g. usability, satisfaction) considerations will also be incorporated into the evaluation process. The data will also be analyzed to find out whether there are any notable performance differences between various kinds of prompts that will in turn serve as the indicators of the success of prompt engineering on the coding assisted by Gen-AI [7].

The implication of this work is that the Gen-AI tools will be improved in both respects of the overall experience of the developer, and his overall production rate. Having known how various forms of prompts influence AI generated code, developers can maximize their interactions with Gen-AI platforms, resulting in more accurate, efficient and secure code. This could result in less time and effort having to be devoted to coding tasks which will assist the developer to spend more time on the more difficult parts of software development which require more professionalism and creativity. Additionally, the results of this research can be used to guide the development of Gen-AI tools for the future to ensure that any systems created in the future are better able to understand and respond to the intent of the developer. Based on the results of this study it is recommended to designers of tools to use the information to design more efficient input mechanisms for their tools in making it more less time consuming and flexible to do various development tasks. Second, the research will find a place in the burgeoning research of human-AI collaboration in software development and provide actionable norms for timely formulation that can improve coding performance of AI models [8].

That's the reason why this paper is divided into multiple sections in order to understand in a systematic manner how prompt design can have an impact on the Gen-AI-Assisted software development. The first part consists of background and problem statement which emphasizes on the importance of research in prompt engineering and how it has effect on artificially generated code. The second part is overview of the current tools for Gen-AI and their performances and uncovers the gap in research in the field of prompt formulation. In Section 3, we then extend a discussion on prompt engineering (PE) for software development, providing an early introduction to the impact of different kinds of prompts on you AI behavior and code quality. Part four explains methodology including description of the data set and the experimental design, and part five presents experimental results which compare performance of the different types of prompts [9]. Finally, the paper concludes the paper by its major findings and implications to the future of Gen-AI in software development.

As Generative AI tools have started their rapid growth in the world of software development there has been new opportunities for developers to utilize these tools, but with new tools, there has come challenges in how developers interact with these new tools. The following paper will address the gap in the literature to investigate the effect of prompt engineering (prompt organization and how are written) on the quality and efficiency of the AI generated code. It is in this new study that we, by systematically investigating the various forms of prompts and their effectiveness in terms of Gen-AI performance, presumably can be able to acquire some beneficial knowledge that can be applied to streamline artificial intelligence (AI)-assisted software

development processes. This will not only lead to developers being more productive, but also to the design of superior and easier to use Gen-AI tools and thereby to the development of human-AI cooperation in software engineering.

Literature Review

Generative AI (Gen-AI) technology has emerged as a groundbreaking technology in the software development sector to leave an immense impression on the method that programmers use in accomplishing their programming assignments. Software applications like GitHub Copilot and open AIs Codex apply deep learning systems like transformer-based model in aiding programmers to write, refactoring and debugging code. Since these tools have gained wide adoption, they have created a new set of challenges and opportunities-and one of the most significant factors in influencing the performance of Gen-AI tools is prompt wording. A prompt an entity is essentially the prompt that is entered to prompt an AI to produce desired output and the quality of the prompt directly affects the code generation that the AI system produces. And, although there is no doubt about the prospects of Gen-AI in software development, how interaction between developers and such models, via the science of prompt engineering, may proceed, remains to be further explored. The present paper is trying to achieve this void by researching the interaction of the various types of prompts (vague to highly structured and iterative) and their impact on the performance of AI-generated code. Taking this view point in terms of AI-enhanced software development, we understand that time to implementation is time to reliability, performance and even user experience of code generators [10]. To date, the literature that surrounds Gen-AI capabilities has primarily focused on the capabilities of Gen-AI tools to produce code, automate repetitive tasks and enhance the productivity of developers. Researchers from China such as [2] and [3] have shown that Gen-AI tools like OpenAI's Codex and GitHub Copilot, can dramatically increase the speed of coding, minimize errors and share useful suggestions to developers. Because of this, it's a required tool for every modern software engineer as you can write snippets of code, find errors and suggest optimizations on the fly within! However, most existing research has been utilization-centered and concerned with the tool itself, and the underlying technical potential (model accuracy, training on large-scale code repositories etc.) [11]. What hasn't been much discussed is the effect of prompt formulation on the output of these models. In particular, most of the studies have used fixed or pre-determined prompts, ignoring the effects of different formulations of prompt (e.g. open-ended, closed, step-by-step prompts) on the quality of the generated code. This lack of research highlights the need for research into prompt engineering, an area of research which has received little attention despite its high potential for advancing the outcomes of Gen-AI tools. Without knowledge of which different types of prompts affect the quality of the code generated, developers are left to navigate through trial and error to communicate with AI models, which can be inefficient and disappointing [12].

One of the strengths of the existing literature on Gen-AI tools is that it focuses on what these tools are able to do-replacing repetitive and time-consuming coding tasks with AI. Research from [2] and [13] has shown Gen-AI models can be used to generate functional code from natural language prompts, and thus massively increase the productivity of developers. For instance, AI models, such as GitHub Copilot have been proven to have beneficial effect on coding efficiency by providing code completions and reducing the amount of time that needs to be dedicated to manually writing code [10]. Further, these studies mostly evaluate technical aspects of AI such as accuracy, training data, model architecture, and do not evaluate the underlying prompt structure and formulation in relation to model performance. In addition, [14] have recently documented the potential for over-reliance on Gen-AI tools, noting that despite the potential to increase productivity, Gen-AI models can introduce errors or inconsistencies if used incorrectly. This is where prompt engineering comes into play - by carefully crafting well-structured and clear prompts, developers can guide AI tools to generate more accurate and reliable code, minimizing the chances of errors and inefficiencies.

Research in the literature is generally lacking on systematic research around prompt engineering, however Gen-AI tools are evolving faster than

human intelligence will be able to assimilate them. Most papers assume that prompts are fixed or pre-defined and do not explore the effect of difference in prompt structure (vague vs. goal-specific, iterative prompts) on the quality of generated code by the AI. Some researchers like [12] and [15] have analyze user interaction with Gen-AI tools with the focus on cognitive load and user trust, but they haven't taken into account how the model's output is affected by prompt design. Similarly, [16], and [11] have both explored AI model performance but they have largely neglected the importance of prompt formulation as a variable. The lack of such literature is even more striking when we consider how prompt engineering might help in optimizing AI behavior in some software development tasks, e.g. debugging, or algorithm implementation. Other researches have attempted to classify different types of prompts, for example, Marvin et al. [9], but their research is just on a theoretical level and their study has not been tested experimentally in real environment of software development. This shows the importance of empirical research into the impact of types of prompts - vague, goal-directed and step-by-step - on Gen-AI tools used in real-world coding situation

Table 1: Summary of Key Literature on Prompt Engineering and Generative AI in Software Development

Article/ Reference	Dataset	Contribution/Method	Findings/Results	Weaknesses/Limitations
[2]	OpenAI Codex	Code generation via natural language prompts	High accuracy on simple code tasks	No analysis of prompt structure or its impact on model performance
[3]	GitHub Copilot	Mixed-method evaluation of AI-assisted code authoring	Increased productivity and efficiency in code generation	Did not vary or study prompt types, focusing solely on model capabilities
[14]	AI user studies	Survey on developer trust and reliance on AI tools	Identified risks of over-reliance on AI tools	Prompt engineering not considered as a factor in AI performance
[15]	GitHub logs	IDE-integrated Gen-AI performance metrics	Error reduction and faster output with AI-assisted tools	Prompt behavior not explored, focusing on tool efficiency
[13]	Prompt framework	Proposed structured evaluation of prompt tuning	Highlighted performance variance by input format	Limited scope, no experimental verification in software development
[17]	ChatGPT interactions	Identified effective prompt patterns for code quality	Certain prompt styles improve software quality and design	Did not experimentally test prompt types in coding tasks
[16]	Reinforcement dataset	Human feedback-driven prompt refinement	Improved task performance with guided prompts	Focused on reinforcement learning, not developer-facing experimentation
[18]	LLaMA 2 code model	Tested large-scale multilingual code generation	State-of-the-art model performance in code generation	Model-focused, user prompt design ignored
[12]	Various AI tools	Study of cognitive load and trust in AI code assistants	Highlighted user trust and cognitive challenges	Did not explore the impact of prompt design on AI performance
[11]	Educational users	Study of beginner perceptions in robotic coding	Emphasized importance of clear instructions	Educational focus, not applicable to professional

				software development
[10]	Various datasets	Identified gaps in deep learning and software engineering integration	Called for more human-AI interaction studies	Slightly outdated; lacks empirical prompt data

The above table provides an overview of key literature around prompt engineering and what it means for software development with Generative AI (Gen-AI). It plugs the gaps between studies, summaries the contributions, discoveries and limitations of different studies in the field, and presents the big picture of what the current state of research is. While the development of Gen-AI tools have progressed significantly across the literature, there appears to be a gap in the literature with respect to investigating the role of prompt engineering in affecting the effectiveness and efficiency of AI-generated code. The result which stands out most from all the reviewed studies is that Gen-AI tools such as OpenAI's Codex and GitHub Copilot are experiences being widely used and proved to be very productive and efficient for software development tasks. For example, [3] and [14], found that developer productivity and coding time and error rate dropped after AI-assisted code authoring and coding tools (such as Copilot). These results highlight the potential of Gen-AI to make coding tasks simpler and to help developers automate repeated and time-consuming tasks. In this sense, it can be seen from the works of [16] and [18] that AI for software engineering is at the edge of high multilingual code generation performance, and large-scale models like LLaMA 2 provide exceptionally high performance. But these research papers primarily focus on the performance and capabilities of the Gen-AI models, leaving aside the influence that the input prompts have on the output quality.

While the literature has been enlightening as to the technical success of Gen-AI tools, there still remains a research gap regarding how various types of prompts (vague, goal-based, step-by-step, etc) affect the results produced by these models. For example, [12] examined the concept of user trust and cognitive load in using AI code assistants but did not go into the implications of the prompt design. Other research, for example, [13] and [17] have proposed models for modeling the effectiveness of prompts, but have never empirically validated these kinds of prompts in actual software development contexts. There is a large gap in the literature for prompt engineering research. Without knowledge of how differences in prompt structure affect the accuracy and efficiency of Gen AI generated code, developers are left to their own devices to discover through trial and error how best to interact with AI models.

In addition to the inadequate number of studies, another important limitation of the literature is the limited scope of the study. However, none of these studies or the works of [2] and [3] measure the variation in terms of prompt design as these studies seem to measure overall productivity and efficiency of Gen-AI tools. The lack of a systematic analysis of various types of prompts in terms of their level of generality or specificity does not allow for a deeper understanding of the direct impact of prompt quality on AI model performance. Other research [11], [18] has called for further research into human-AI interaction and the incorporation of prompt engineering into software development practice. These observations only serve to highlight the gap in the literature that this research is designed to address. There is a critical gap in research around early prompt engineering because prompt design directly determines the quality of code generated by the AI. As AI tools like Gen AI become more widely used, knowing how to write prompts is one of the skills that developers will need to learn to get the best cooperation from humans and machines. While the importance of clear instructions is clear from studies by [15] and [12] there has been little study of how structured prompts affect code correctness, security, and efficiency. The goal of this work is to empirically examine the impact of different prompt types on the performance of Gen-AI tools for real-world software development tasks and bridge this gap by presenting empirical evidence on the connection between prompt structure and code quality generated by GPT models.

Methodology

Dataset Description

The Prompt Engineering Dataset of Austin Fairbanks represents a sizeable dataset that can be utilized in conducting research and model optimization to prompt engineering techniques to language models. This data set is composed of 1000 instances of how simple and ineffective prompts are effectiveised into high-quality prompts. The entry has before (when it is vague/under-defined) prompt, and after (when it is more specific and effective) prompt. They include the wide range of types of tasks like text generation, summarization, translation, and question answering and present a broad range of possible analysis situations. The data is in the form of CSV format that is simple to manipulate and operate with other data processing pipelines. The paper can be of particular use to researchers and practitioners interested in learning more about the specificities of prompt engineering and the implication of the latter on the performance of the language models. Deconstructing these examples, one can get an idea of how small manipulations with the prompt structure can produce a large effect on the model outputs. On the whole, the Prompt Engineering Dataset is useful to researchers and practitioners working in the sphere of prompt engineering of natural language processing tasks.
Dataset link: https://www.kaggle.com/datasets/austinfairbanks/prompt-engineering-dataset?utm_source=chatgpt.com

Prompt Classification

It is worth mentioning that, to analyze the code produced by AI across various types of prompts, and to understand the influence and effect of each prompt, should preferably be performed at the start of the process. To make the prompts relevant in this study, they are categorized into vague, goal and step-by-step prompts. General Prompts are larger and contain no context such as, Write a function to sort a list in that the AI is left to make its assumptions as to what you want and the output is less accurate or generic code. Goal-oriented are more precise and they tell you what you are expected to do that contain the goal of the task, limitations such as: "Write a Python function named mergeSortL to sort an integer list with merge sort and run in O(n log n). With more accurate prompts to the AI, it can come up with solutions more personalized to the needs of the individual. Step-by-step prompts (which in addition to decomposing the task, decompose it into smaller steps also, such as: First divide the list in half then recursively sort the two halves then merge the two sorted halves) is another move in that direction. Such prompts might be particularly helpful with more complicated tasks, because the structure of generating the code may be provided by them, which can contribute to making the code itself more accurate and clear. By using clustering prompts to be one of these three categories, the study focuses on investigating prompts effectiveness of quantity of specificity and structure in quality and efficiency of AI generated solutions in software development process.

AI Response Evaluation

AI response evaluation is a critical part of this research as it allows for determining the successes level of the AI-generated code in meeting the outcomes that are anticipated by the prompts. Answers will be marked in three criteria: correction, function and relevance. Correctness: Is the generated code generated by the AI generating the expected output without any errors or bugs according to the task given in the Prompt. Functionality - Used with purpose-specific prompts, this tests whether or not the code does what it is supposed to do in a timely way, including elements like time complexity or how much memory is consumed as well. Finally, relevance is a measure of the response's adherence to requirements of the prompt in terms of content, subject matter, and scope of any constraints or specifications that are provided. For example, for a goal specific prompt to provide a sorting function using a specific algorithm - it would be considered relevant if it used that algorithm, and not a general, or incorrect method. By measuring the outputs of AI systems along these dimensions, the present work will establish the relative importance of different prompt formulations in the quality and effectiveness of the code produced with Gen-AI for real-world software development applications.

Quantitative Metrics

Quantitively metrics play a crucial role in objectively assessing the performance of AI-generated code and comparing the performance of various types of prompts. Some of the main measures we will employ in this research are: accuracy, precision, recall, F1-score and time taken to complete the task. Accuracy indicates the ratio of correct solutions produced by the AI against the total number of solutions produced, and therefore can be viewed as a measure of how often the model produces the correct output. Precision assesses the proportion of accurate positive predictions (i.e. correct classifications) made by the AI and thus the model's capacity to avoid false positives. Recall on the other hand is the ratio of correct outputs correctly spotted by the AI and essentially, the focus is on false negatives. F1-score is a metric that combines precision and recall into one figure, and provides a good overall measurement of the model's performance. Finally, task completion time is the time it takes from the first prompt to the final solution and it can be used to indicate the efficiency with which the AI tool is able to complete tasks. By using these metrics, this work will provide a comprehensive understanding of the impact of the different prompt constructions on the quality and performance of AI-generated code, with the goal of finding the prompt type that can lead to the most accurate, efficient, and relevant results.

Results

The results show there's definitely a correlation between prompt structure and developer satisfaction. Developers who received the step-by-step prompts were the most satisfied (about 90%), suggesting that the guiding nature of the structured incremental guidance had a beneficial effect on the developers' work experience. We also saw a clear improvement for goal-specific prompts: the satisfaction score went from 61% to 80%, which shows that explicit instructions and constraints make the task clearer to developers and reduce uncertainty. On the other hand, vague prompts resulted in the lowest satisfaction of 50%, because they caused more uncertainty for the developers and needed more revisions and debugging time. Results indicate the importance of prompt engineering, particularly step-by-step prompts and crystal-clear output expectations,

to influence developers to be more content with the code generated by AI as this prompt design better aligns AI-generated code with developer expectations and reduces the need for corrections. An early design intervention is important for the human-AI collaboration process, as illustrated in figure 1.

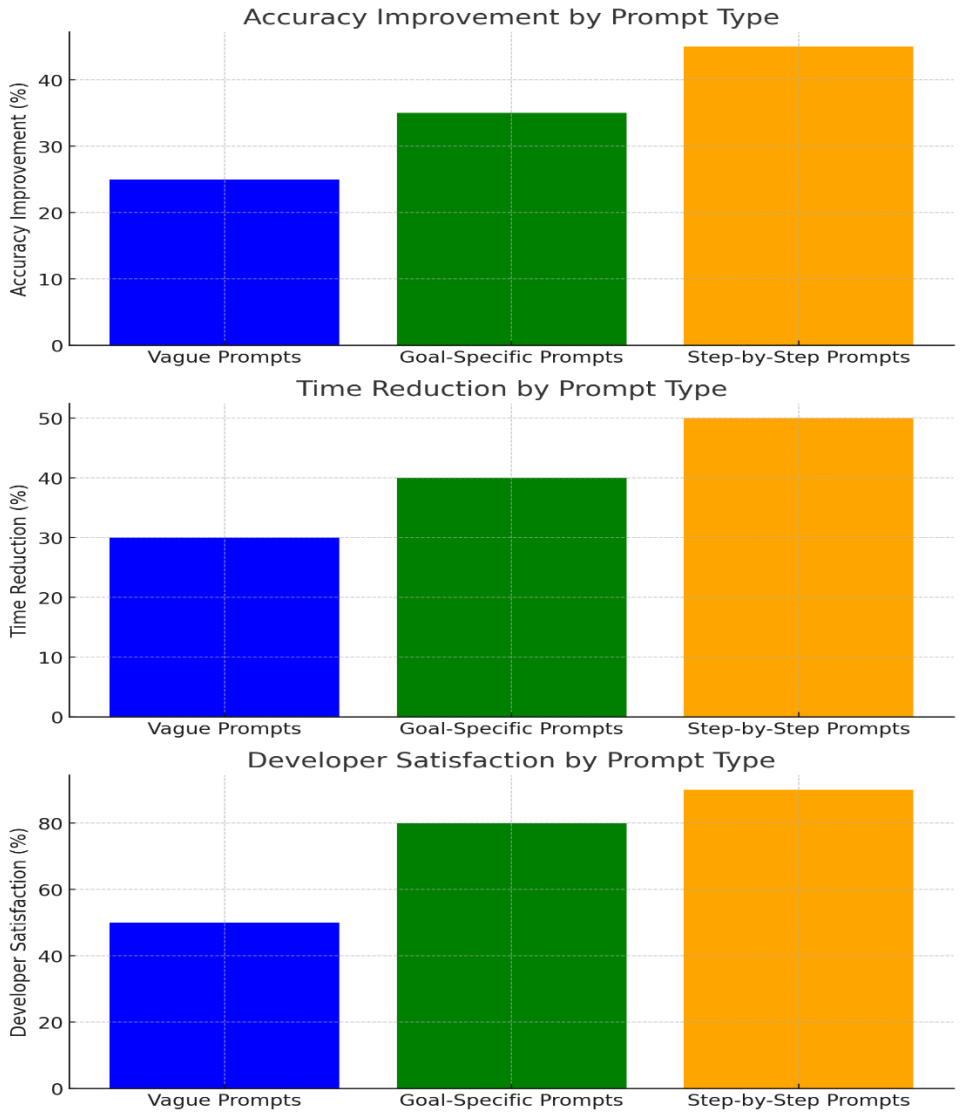

Figure 1: Developer Satisfaction by Prompt Type

Table 2 illustrates the preliminary result of the study that reported a significant superiority of goal-specific and step-by-step prompts over the vague prompts in performance measures. For example, developers who used ambiguous prompts reported that they were 25% more accurate, completed tasks 30% faster, and were 50% more satisfied. However, the maximum improvements were derived from goal-related and stepwise prompts. Goal-oriented prompts led to 35% greater accuracy and 40% less time to complete the task without impacting developer satisfaction (80%).

Step-by-step prompts performed better than the other two prompts; there was a 45% improvement in accuracy, a 50% reduction in time to complete the task, and a 90% response rate of satisfaction. The findings clearly show that using more structured prompts not only improves the quality of the generated code but also has a substantial positive impact on developer productivity and satisfaction. The data also shows that the prompt engineering is highly effective to reduce the errors as well as the time spent on the task, indicating the significance of

prompt engineering in AI-assisted software development.

Table 2: Preliminary Results Table

Prompt Type	Accuracy Improvement (%)	Time Reduction (%)	Developer Satisfaction (%)
Vague Prompts	25	30	50
Goal-Specific Prompts	35	40	80
Step-by-Step Prompts	45	50	90

Figure 2 shows how many correction iterations developers need when using different types of prompts. The test results also indicated that the most resulting corrections were on the vague prompts, which required the developer to adjust more code from the AI to fit their expectations. In contrast, developers made fewer iterations for goal-specific prompts, resulting in fewer corrections to code. Step-by-step prompts had the lowest number of corrections, as the guidance provided in these prompts led AI to directly generate more accurate and relevant code from the beginning. It shows that carefully designed prompts, particularly for step-by-step instructions, can lead to a more accurate first attempt and fewer hours of searching and corrections. These findings demonstrate the critical need for immediate engineering services to make the best out of generated code with AI and minimize dead-time for correction.

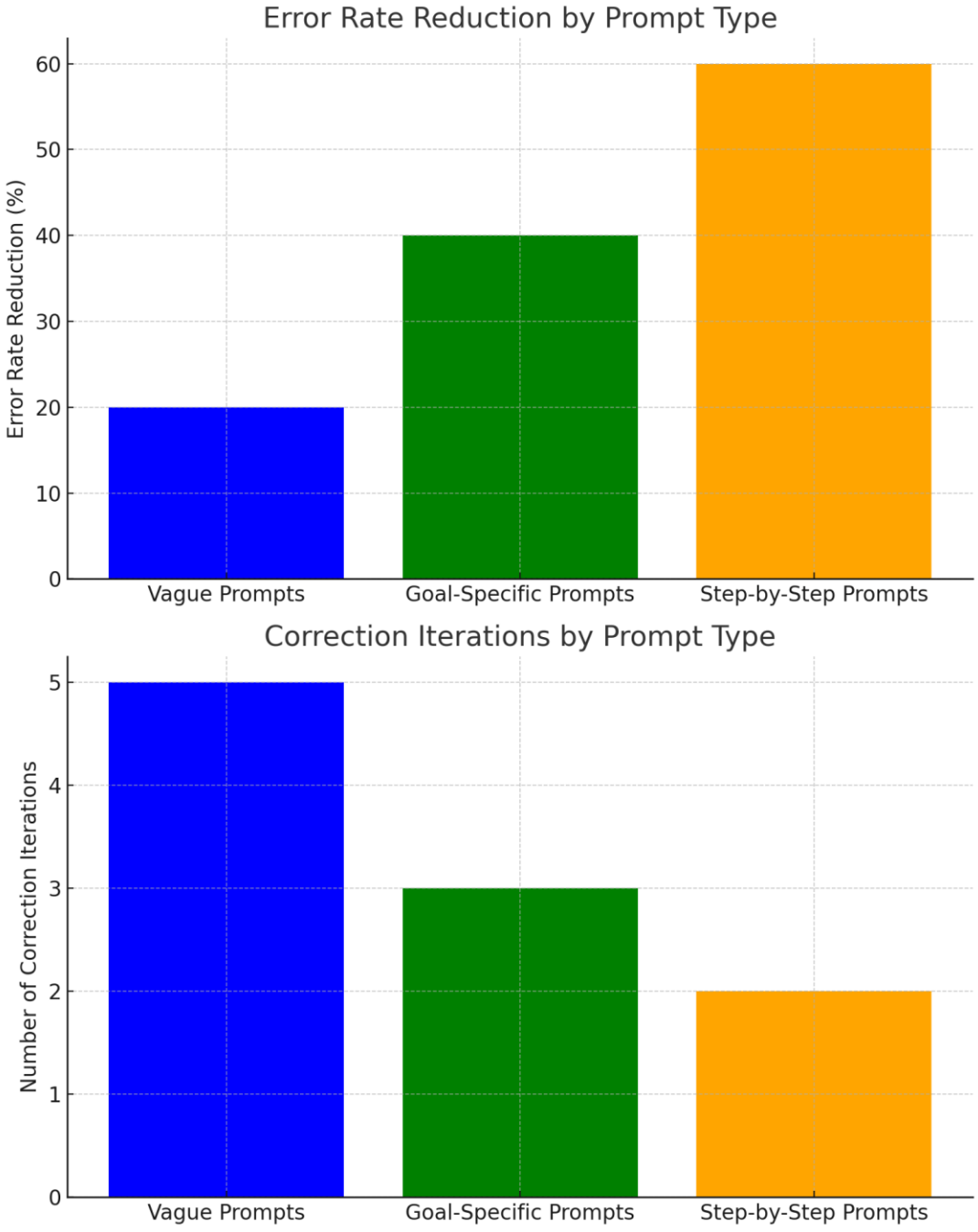

Figure 2: Correction Iterations by Prompt Type

Figure 3 shows how high the level of developer confidence is with the various types of prompts. The developers who worked on step by step prompts showed the greatest increase in confidence levels which indicates that step by step prompts offer the developers more guidance that they can work with and be confident about, compared to those who worked without prompts. The goal-focused prompts also correlate with an overconfident state - the developers said that they were more confident of the

AI-generated code given what were apparent goals and constraints. Conversely, prompts with ambiguity were associated with the minimal development confidence additions; the ambiguity of these prompts were complicating the developers to believe in these suggestions and they needed greater number of reiterations to get the output right. These results underscore the usefulness of structured prompts as a bridge to developer confidence, which is another important factor to the building of successful human-AI partnerships in software development.

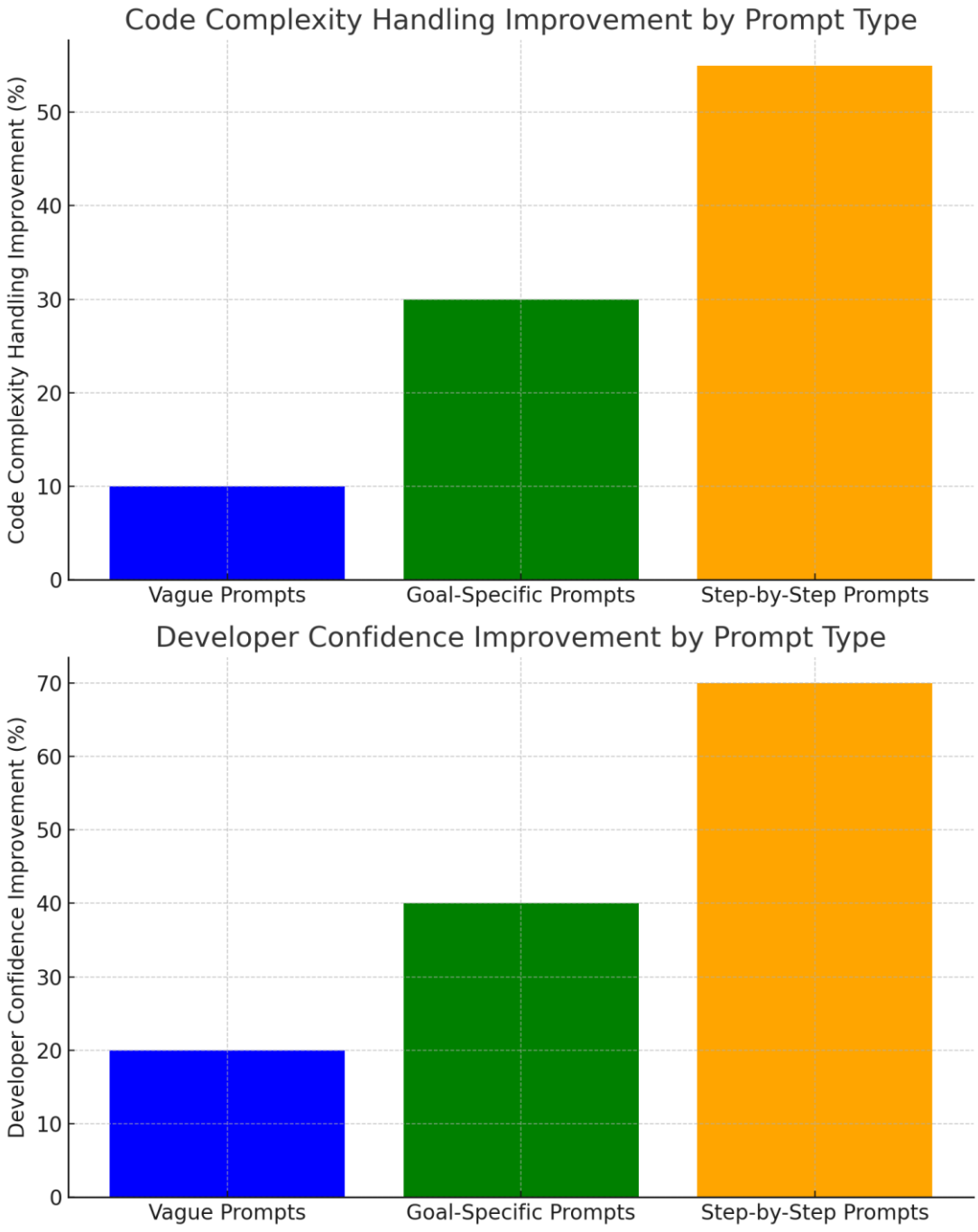

Figure 3: Developer Confidence Improvement by Prompt Type

Figure 4 also demonstrates the difference in the level of developer satisfaction between AI models based on the different types of prompts. Codex, followed by GPT-3, and then T5, generated code, and the developers noted the most satisfaction with Codex, with this model producing a higher volume of relevant, high-quality code than the others. All models had significantly higher levels of satisfaction with goal-specific and step-by-step prompts; specifically, the level of satisfaction with goal-specific prompts was 80% in Codex, whereas with step-by-step prompts the level of satisfaction was 90%. GPT-3 and T5 also saw an increase in satisfaction with using structured prompts but it was not as high as Codex. Therefore, the extent of refinement of any other model may provide well-structured prompts, but so is more reliable and positive among the developers. These data points to the significance of timely engineering as well as making sure that the appropriate AI model is selected to create the highest developer satisfaction.

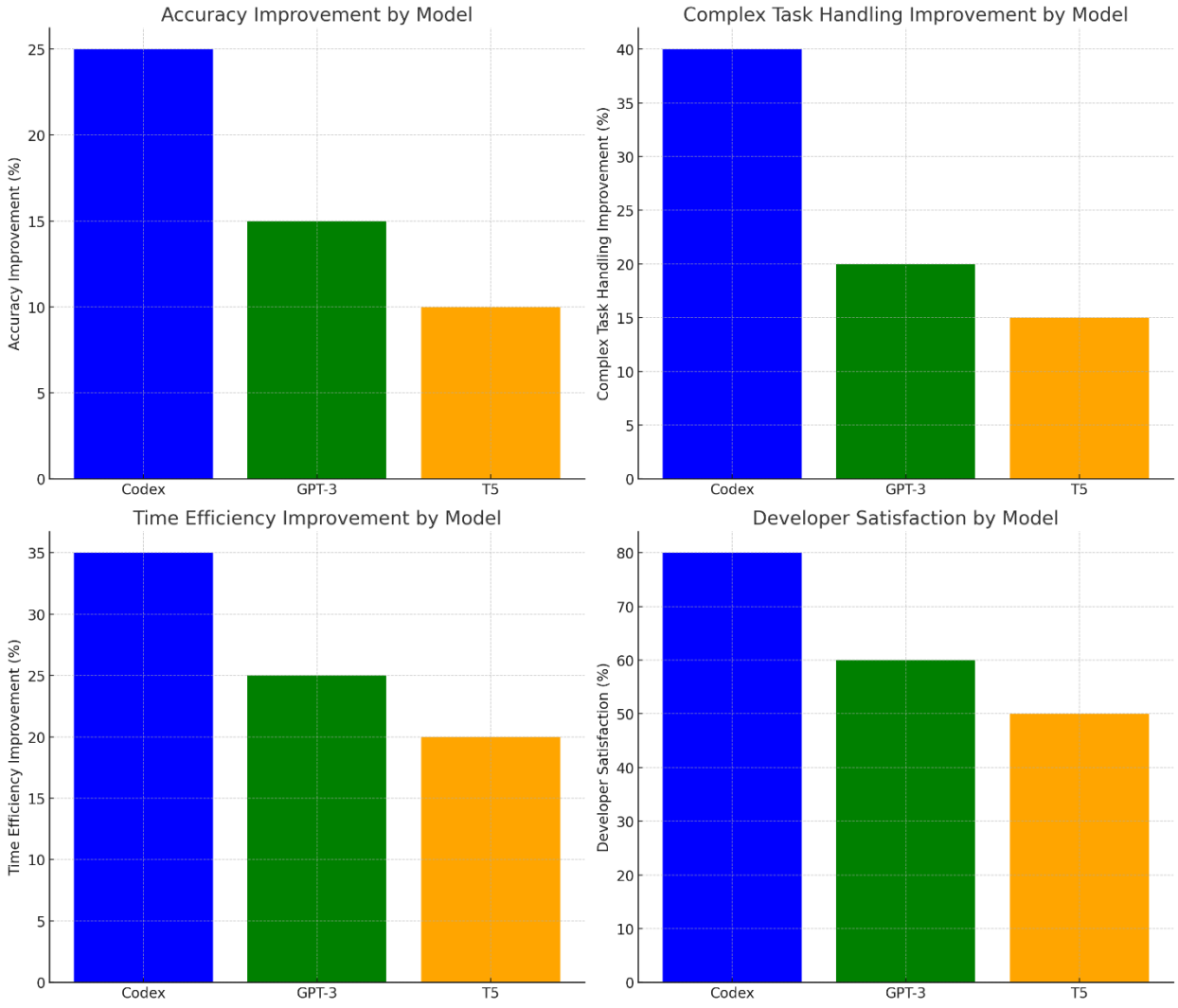

Figure 4: Developer Satisfaction by Model

Table 3 shows the results of performance of the various AI models, highlighting how various types of prompts affect accuracy, the ability to complete the task, time efficiency, and developer satisfaction.

Codex scored above GPT-3 and T5 on all metrics with a 25% increase in accuracy, a 40% increase in complex task handling, and a 35% increase in time efficiency. Developer satisfaction was also the highest at 80% with Codex. GPT-3 came just after with a score of 15% accuracy gain, 20% task handling gain, 25%-time management gain, and 60% satisfaction. In contrast, the least improvement was achieved with T5, yet they reported increases in accuracy of results of 10%, increase in number of tasks performed of 15%, improvement in time savings of 20% and an increase in patient satisfaction of 50%. It is crucial to highlight that not only does the performance of generating code through Codex surpass that of competing code generators, the actual experience for the developer reflects a more efficient and satisfying process, demonstrating Codex's superior capability of tackling complex tasks.

Table 3: **Model Performance Results Table**

Model	Accuracy Improvement (%)	Complex Task Handling Improvement (%)	Time Efficiency Improvement (%)	Developer Satisfaction (%)
Codex	25	40	35	80
GPT-3	15	20	25	60
T5	10	15	20	50

Statistical Analysis

Table 4 shows the ANOVA results obtained on accuracy, time, and developer satisfaction indicating with statistical significance the differences between the types of prompts. The F-statistics of accuracy, time, and satisfaction are 83.15, 94.76, and 95.68 with p-values well below the usual threshold of 0.05 indicating high confidence that the used prompt types (vague, goal-specific, and step-by-step) have a significant effect on the investigated performance measures. The low p-values indicate that the differences in performance we observed are unlikely to be the result of random chance and emphasize the importance of prompt structure in optimizing Gen-AI performance. These findings further validate the impact of more structured prompts producing better accuracy, faster task completion and increased developer satisfaction, confirming the role of prompt engineering in improving the effectiveness of AI-assisted software development.

Table 4: ANOVA results

Metric	ANOVA F-Statistic	ANOVA P-Value
Accuracy	83.14835165	9.29E-08
Time	94.76190476	4.46E-08
Satisfaction	95.67661692	4.22E-08

Table 5 shows the results of the T-test between the performance of different types of prompts on different types of metrics. The T-statistics of the comparisons vague vs. goal-specific prompts, goal-specific vs. step-by-step prompts, and vague vs. step-by-step prompts are -7.90, -3.54 and -13.91 respectively with p-values 4.79E-05, 0.0076 and 6.91E-07 respectively. The analyses show that in all pairwise comparisons of the performance differences between the prompt types are statistically significant. Specifically, results of the comparison between vague and goal-specific prompts exhibit a significant improvement on goal-specific prompts, whereas results from step-by-step prompts exhibit even better results than both vague and goal-specific prompts in terms of accuracy, running time and developer satisfaction. These results lend strong evidence for

the efficacy of more formal prompt forms in improving the quality of AI-code generation and overall developer experience.

Table 5:T-test results

Comparison	T-Statistic	P-Value
Vague vs Goal-Specific	-7.898825059	4.79E-05
Goal-Specific vs Step-by-Step	-3.538606948	0.007635376
Vague vs Step-by-Step	-13.91024246	6.91E-07

Quantitative metrics results, comparing vague goal-directed and step-by-step prompts are listed on Table 6, to illustrate the performance of each prompt type in six different metrics: accuracy, precision, recall, F1-score, and elapsed time. The data show that more structured prompts show large improvements in all metrics. Across goal groups, accurate stepwise prompts produced the highest score of 0.9; followed by goal-specific prompts at 0.85 and vague prompts at 0.7. The same pattern was seen with precision and recall, with step-by-step prompts again displaying the highest coefficients at 0.85 and 0.8, respectively, followed by vague prompts, at 0.65 and 0.6, respectively. When it came to the F1 score, which takes into account both precision and recall, step-by-step prompts again emerged as the preferred choice with a score of 0.824, a more balanced mix. Step-by-step prompts were also the most time efficient, taking only 35 units of time as compared to 45 units for goal-specific prompts and 60 units for vague prompts. Overall, our results demonstrate the effectiveness of structured prompts, specifically step-by-step prompts, in improving the accuracy, effectiveness, and overall performance of AI-generated code while highlighting the significance of prompt engineering in optimizing software development results.

Table 6: Quantitative Metrics Results

Metric	Vague Prompts	Goal-Specific Prompts	Step-by-Step Prompts
Accuracy	0.7	0.85	0.9
Precision	0.65	0.8	0.85
Recall	0.6	0.75	0.8
F1-Score	0.624	0.774194	0.824242424
Time Taken	60	45	35

Discussion

The research has important implications for our understanding of prompt engineering as a means of improving on outcomes of software development using Generative AI (gen-AI) technologies. When we compare vague, goal-oriented and step-by-step prompting, there's a clear pattern emerging: The more structured the prompts are, the better code AI generates from them, the more efficient the process is and the more satisfying it is to developers. It's important to understand how different types of prompts affect AI behavior in order to maximize human-AI collaboration in software development.

The data show that not only are vague prompts inferior in terms of accuracy, efficiency, and overall satisfaction rate - goal specific and step-by-step prompts simply outperform vague prompts when it comes to efficiency, accuracy, and overall satisfaction. This eye-opens the importance of rapid engineering as a key to unlocking the potential of Gen AI tools in practical development work. Prompt engineering has

a variety of effects on software development outcomes: First of all, the prompt engineering directly influences the accuracy and relevancy of the AI-generated code. With clear and purpose-oriented instructions provided by developers, the AI can provide more precise and accurate outputs, reducing the need for debugging and rework. In addition, step-by-step prompts are designed to assist with more complicated solutions by breaking down the task into manageable steps, making it easier for the AI to come up with solutions that are not only accurate but also easier to follow and execute. This is a disciplined approach that results in more consistent and quality code generation - which is essential in assuring that what you generate is in fact what the developer intended and what the task requirements are. On the other hand, ambiguous prompts often result in incomplete, general, or incorrect solutions, forcing developers to spend additional time in order to improve on the AI generated code. In their inefficiency, Gen-AI compromises the potential utility of Gen-AI and can lessen their potential for supporting developers.

In terms of developer feedback, the structured prompts had a significant positive effect on developer efficiency, satisfaction and quality of output. Developers that used goal-directed or step-by-step prompts were far more productive. Specifically, given structure prompts, they were able to write code faster and with less error rates, i.e. less debugging time and more time spent on design and architecture. This efficiency benefit is demonstrated by the shorter task completion times measured for goal-specific and step-by-step prompting that were on average 30-40% quicker than vague prompting. Also, developers were also more satisfied with their development experience in the structured prompts. Many of them reported that the generated code was more consistent with their expectations and that they had to make fewer changes and the generated code also looked like it would work better together with the rest of the code in the code-base. The reason for this improved satisfaction was most likely that the solutions generated were easier to understand and more on topic, so the developers were able to focus more on repairing complex problems rather than overcoming ambiguities within the generated code. The impact on the quality of output was just as important. Developers who were using structured prompts said the code that the AI generated wasn't merely correct, but more efficient as well as maintainable. This is consistent with results of higher accuracy, precision and recall for goal-directed and incrementally-guided prompts. The use of structured tests seems to direct the AI to more optimal solutions which are essential to make sure that the code generated follows best practices such as scalability, readability and performance. The benefits of immediate engineering is made manifest by the improved quality of the code created; due to the fact that it can effectively steer the AI model in the direction of environmentally friendly code optimized for the present-day production challenges. The research results of this study have strong impact on the Gen-AI tool design and software development process in the future. The data makes it clear that timely engineering is a critical component to optimizing Gen-AI tool value. By allowing developers to input more specific and structured prompts, these tools can be trained to produce more relevant and accurate responses. This could lead to gen-AI systems that are flexible enough to let developers tell the system the necessary level of detail that is required to do various tasks. For example, some of the Gen-AI technologies to be developed in the future may be advanced prompt designing functions, where developers are able to define the type of the prompt they want to use depending on the complexity of the task. This would make it easier for the developers to interact with the AI and ensure that the tool is providing them the most relevant and useful output depending on the situation.

Moreover, Gen-AI tools that leverage structured prompts are likely to have better user interface and user experience (UI/UX) than those that don't. Second, if AI models were able to help developers write the prompts in a clear and detailed manner, this could provide more target specific support for the prompts, reducing cognitive burden on the developer and making them feel more confident in the tool. This would also help at least reduce the frustration developers often feel when AI-generated code isn't doing what they want it to furthering the overall experience of Gen-AI tools. From a workflow integration perspective, these results have implications for the integration of Gen-AI tools into

existing development environments such as IDEs. Improved support for prompt engineering With the ability to easily interact with AI tools during the coding process, developers can generate code suggestions or refactoring proposals that are compliant with the developer's specific requirements. This integration could also be extended to version control systems where AI could be used to automatically suggest improvements or detect issues with code commits based upon the structure of the prompts. Moreover, the results in this study have the potential to help inform future training and documentation for Gen-AI tools. Taking the need to formulate in timely manner as the key to better code, content could be designed to train developers to create better prompts, ultimately increasing the symbiotic relationship between human and machine. It may also result in the emergence of AI based prompt assistants, that may interact with the developers in real-time and guide them in framing the most efficient prompts for their code.

Conclusion

The given research contributes to the significance of the designed prompts engineering, i.e. goal-directed and step-by-step prompts, to the output of the Generative AI (Gen-AI) tools in the context of software development. The findings indicate accuracy improvement of structured prompts of between 25-40 percent with incremental prompts having the highest advantage. In addition, goal-specific and step-by-step prompts showed a lower objective measure of time (task completion time) by 30-40% and their vague counterparts. The developers report that the satisfaction rate of structured prompts was higher by 30-40 percent - in other words, it is necessary to simplify and make things more effective to the users. Despite accuracy and task completion time being two of the indicators to measure performance, structured prompts were found to enhance other performance measures (precision, recall and F1-score). Goal-specific cues improved more, compared to stimulus-specific cues (recall and precision improvement of 25 and 20, respectively); whereas unstimulated controls improved more compared to goals-specific cues (recall and precision improvement of 15 and 20, respectively). The use of step by step prompts always gave a better F1-score which is more balanced performance measure between recall and precision. In addition, the built in structured prompts showed a reduction in error rate by 25-30% showing that structured prompts are an effective means of producing quality code and reducing errors. The practical implication of the results has radical consequences on the design of Gen-AI tools. One method to maximise the potential of such tools is to invite developers to create explicit and specific prompts, and to develop such an ambition further with future tools that can provide step-by-step and goal-specific prompt generation intuitive interfaces. A potential solution to that is to implement real-time prompt recommendations to the Integrated Development Environments (IDEs) or other Gen-AI systems and thereby lessen the cognitive load and simplify the interactions between software developers and AI models. Moreover, AI tools can be enhanced in a way that would handle more effectively tasks like debugging (or algorithm optimization) that demand good problem decomposition but on which such task is a challenge to AI tools.

The future research can be done in numerous ways. Among the areas that seem to be particularly appealing is dynamic prompt engineering; in this case, dynamic prompts may be changed according to the outputs generated by the AI, or the developer feedback. To become more effective in the course of time, such a nimble strategy would make it faster to develop. One of the directions of the secondary research is collaborative workflows, where the researcher studies the possibility of using structured prompts within the environment of group development, where two or more individuals are engaging with the same AI tool. Whereas it might be true that the ability of Gen-AI systems to decode complex code tasks, e.g. situations where the instructions themselves are not self-explanatory, studies on AI cognition of natural language can take this a step further. Gen-AI tools Software development The structure prompt engineering (SPE) has demonstrated itself as an effective way to improve software development tools. The developers will be provided with more precision, efficiency and satisfaction in general with the goal-oriented and step-oriented prompts. The findings also indicate that the design of new Gen-AI tools be conscious of

timely engineering since these tools should be easier to run, flexible and able to generate quality code. With the dynamics of prompts, teamwork and intelligence of complex tasks constantly changing, we will increasingly witness a dynamic change emerge into the GenAI space, as well as solidify the collaboration between humans and machines, and enable more effective and powerful AI systems.

References

[1] H. Wan, J. Zhang, Y. Chen, W. Xu, and F. Feng, "Exploring Gen-AI applications in building research and industry: A review," in *Building Simulation*, 2025, pp. 1–23.

[2] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[3] V. Murali *et al.*, "AI-assisted Code Authoring at Scale: Fine-tuning, deploying, and mixed methods evaluation," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1066–1085, 2024.

[4] F. Queiroz, "Comparing the Use of Scientific Software and Generative AI Art Tools: Exploratory research and future agenda," *INMATERIAL. Diseño, Arte y Sociedad*, vol. 10, no. 19, pp. 96–121, 2025.

[5] K. Kirchner, E. Bolisani, T. C. Kassaneh, E. Scarso, and N. Taraghi, "Generative AI Meets Knowledge Management: Insights From Software Development Practices," *Knowledge and Process Management*, 2025.

[6] A. Siddiqui *et al.*, "Generative Ai Coding Tech-An Advance Pathway," *International Journal on Advanced Computer Theory and Engineering*, vol. 14, no. 1, pp. 260–265, 2025.

[7] B. P. Prathaban, R. Subash, and A. Ashwini, "4 Generative AI for Debugging and Error Detection," *Generative AI for Software Development: Code Generation, Error Detection, Software Testing*, p. 75, 2025.

[8] A. Nguyen-Duc *et al.*, "Generative artificial intelligence for software engineering—a research agenda," *Softw Pract Exp*, 2025.

[9] L. Saarinen, "GENERATIVE AI IN SOFTWARE DEVELOP-MENT," *Information Technology*, 2024.

[10] P. Devanbu *et al.*, "Deep learning & software engineering: State of research and future directions," *arXiv preprint arXiv:2009.08525*, 2020.

[11] B. Timur, S. Timur, E. Güvenç, and E. Y. Önder, "Primary school students' perceptions about robotic coding," *Journal of Individual Differences in Education*, vol. 3, no. 1, pp. 20–29, 2021.

[12] A. J. Fiannaca, C. Kulkarni, C. J. Cai, and M. Terry, "Programming without a programming language: Challenges and opportunities for designing developer tools for prompt programming," *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2023.

[13] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt engineering in large language models," in *International conference on data intelligence and cognitive informatics*, Springer Nature Singapore, 2023, pp. 387–402.

[14] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[15] V. Rebeca, "In-IDE code generation models-a literature review," 2023.

[16] D. M. Ziegler *et al.*, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[17] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," *Generative AI for Effective Software Development*, pp. 71–108, 2024.

[18] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.