

Article

Exploring the Impact of LLM Prompting on Students' Learning

Murimo Bethel Mutanga ^{*}, Jotham Msane, Thaddeus Ndimiso Mndaweni, Bongokuhle Brightman Hlongwane and Neliswa Ziyanda Ngcobo

Department of Information and Communication Technology, Mangosuthu University of Technology, Durban 4026, South Africa; msane@mut.ac.za (J.M.); 22116979@live.mut.ac.za (T.N.M.); 22109577@live.mut.ac.za (B.B.H.); 22117916@live.mut.ac.za (N.Z.N.)

* Correspondence: mutangamb@mut.ac.za

Abstract

Integrating large language models (LLMs) into higher education, particularly in programming education, reshapes how students interact with learning materials and develop coding skills. However, while the general utility of LLMs like ChatGPT, Gemini, and Claude has been acknowledged, a critical gap exists in understanding how specific prompting strategies influence student learning outcomes. This issue is significant in the context of programming education, where problem-solving, critical thinking, and conceptual understanding are essential yet complex cognitive skills. Although prior research has classified prompting behaviors, it has largely failed to assess their impact on actual learning. To address this gap, we explored how IT students employ various prompting strategies when engaging with LLMs during programming tasks. A mixed-methods approach was adopted, primarily qualitative and supported by basic quantitative analysis, to examine 842 prompts generated by 140 students across four core software development modules. The results revealed five dominant prompting strategies, which varied significantly in how they facilitated learning. Our findings suggest that prompting strategies significantly shape how students interact with LLMs and influence the depth of their learning.

Keywords: large language models (LLMs); prompt engineering; programming education; ChatGPT; student learning behavior; artificial intelligence in education



Academic Editor: Carlos
Hervás-Gómez

Received: 29 May 2025

Revised: 19 June 2025

Accepted: 24 June 2025

Published: 27 June 2025

Citation: Mutanga, M.B.; Msane, J.; Mndaweni, T.N.; Hlongwane, B.B.; Ngcobo, N.Z. Exploring the Impact of LLM Prompting on Students' Learning. *Trends High. Educ.* **2025**, *4*, 31. <https://doi.org/10.3390/higheredu4030031>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, intelligent systems capable of understanding and generating human-like content have taken center stage, driving a wave of transformation across different fields. These advanced models are not only revolutionizing how we interact with computers but also redefining creativity, problem-solving, and automation. From generating text, images, and videos to engaging in natural conversations, translating languages, and even debugging complex code, their capabilities mark a significant shift in the digital landscape. As these systems continue to evolve, they are unlocking groundbreaking possibilities, reshaping industries, and pushing the boundaries of what machines can achieve.

According to [1], the most widely used large language models (LLMs) today include ChatGPT, Gemini, Claude, Mistral, and Llama, with ChatGPT remaining the most dominant overall. However, ref. [2], found that students in the programming field also frequently use tools like Chegg and Bing Chat (now known as Microsoft Copilot). These two platforms, while not consistently ranked among the top general-purpose LLMs, are often favored for their academic focus and support in educational contexts.

LLMs have significantly simplified various aspects of modern life, including reducing complexity in the field of programming [3]. For instance, a study by [4] found that LLMs have encouraged some students to become active questioners by frequently initiating interactions with the tool and seeking information to deepen their understanding. In contrast, other students preferred a more passive approach, engaging minimally and choosing instead to observe and process information; these students are referred to as silent listeners. While the study provides valuable insight into how students interact with AI agents, it does not evaluate the effectiveness of these interactions on learning outcomes.

For many IT students, AI tools serve as active tutors, offering instant explanations of programming concepts and code snippets. Additionally, this immediate feedback loop enables students to debug errors much faster, understand complicated syntax, and complete tasks efficiently. LLMs also provide support to students who may struggle with language barriers or lack access to an actual tutor [5]. Despite this convenience, it may come with a cost, in which students may start to rely too much on the tool, utilizing it for completing tasks without fully engaging in the learning process [6–8].

This over-reliance would become a hindrance to developing core problem-solving skills such as debugging logic errors and the comprehension of algorithm design on their own [6].

The accessibility of LLMs has contributed to a rise in academic dishonesty, as students are increasingly using AI-generated content to complete assignments and exams, often presenting it as their own work [9]. However, ref. [10] found that integrating LLM tools into students' collaborative programming improved their computational thinking and reduced cognitive load. In contrast, ref. [11] observed that over-reliance on LLM tools for tasks requiring independent cognitive effort has significantly impacted academic performance, resulting in only surface-level understanding for some students.

The growing dependence on LLMs among IT students is influenced by several factors. One major contributor is the increasing complexity of programming tasks, as assignments become more intricate, students often struggle to develop solutions independently. In such cases, LLM tools become an appealing alternative, offering quick code generation, debugging support, and step-by-step assistance, which helps manage these challenges efficiently [12]. Additionally, ref. [13] emphasizes that many students face conceptual gaps in their understanding of programming. To bridge these gaps, they often turn to LLMs for explanations and ready-made solutions. While these tools can support learning, excessive reliance may lead to superficial understanding, as students might use AI-generated code without fully understanding the underlying logic or principles they are expected to master. It is thus crucial for students to understand the best ways of using LLM tools. A key aspect of effectively using LLM tools is prompting. Good prompting techniques often result in the desired output from LLM tools.

Prompting first emerged in natural language processing (NLP) tasks in 2018 with a single dataset and was further refined in 2021 with multiple datasets. By 2022, text-to-code and code-to-code techniques were introduced and have continued to evolve. The text-to-code approach involves describing programming problems in natural language, while the code-to-code strategy uses partial code snippets or hints to help ChatGPT generate complete solutions.

Students' prompting strategies play a crucial role in how they use LLMs for programming tasks. Well-structured prompts lead to more accurate responses, making learning and solving coding problems easier for students. As LLMs become more common in education, understanding how to prompt them is essential for maximizing their benefits. There has been extensive research examining how students formulate prompts when interacting with

large language models, aiming to identify strategies that optimise response quality and support deeper learning.

The work reported in [14] presented a qualitative analysis of students' prompts, identifying three distinct prompting strategies. Moreover, their research also uncovered additional prompting techniques that students use when interacting with ChatGPT. One such strategy is Single Copy & Paste Prompting, where a student copies a question from the problem statement and pastes it directly into the ChatGPT prompt field without modification. The student then adopts the response generated by ChatGPT and submits it to the Moodle quiz without any further revisions. Another strategy is Single Reformulated Prompting, in which a student rephrases the question either partially or entirely in their own words before prompting ChatGPT. The student then uses the AI-generated response and submits it to the Moodle quiz without making any changes. Similarly, Multiple-Question Prompting occurs when a student asks ChatGPT multiple questions before selecting and using the response provided by the model.

While existing studies have begun to explore how students interact with LLMs during programming tasks and have categorized various prompting strategies, such as copy-paste, reformulation, and iterative prompting there remains a limited understanding of how these strategies influence learning and skill development. Much of the current literature is descriptive, focusing on how students use these tools, rather than evaluative, assessing whether different prompting behaviors result in meaningful improvements in programming comprehension, problem-solving ability, or academic performance. Consequently, there is a need for deeper, outcome-focused research that examines whether and how prompting strategies used by IT students with LLM tools directly impact their learning. This study, therefore, seeks to investigate the prompting techniques IT students employ when using LLMs and evaluate the possible impact on their learning.

2. Materials and Methods

2.1. Research Design

This study followed a qualitative research design with minor elements of quantitative analysis. The qualitative approach was best suited because it allowed us to explore how students interacted with Large Language Models when completing programming tasks. Our focus was on understanding the students' intentions, thought processes, and learning behavior through their prompts. Although we counted the number of prompts to identify common patterns, the main goal was to examine the nature and depth of these prompts. To do this effectively, we needed a qualitative lens that could help us interpret not just what students asked, but why they asked it. This mixed approach of using both qualitative and quantitative elements was inspired by three similar research papers. Ref. [3] explored the use of LLMs in programming tasks by combining qualitative analysis of how students interacted with the tools and quantitative analysis of prompt frequency and task performance, which demonstrated how both methods can be used together to gain a complete understanding of learning behavior. The work in [4] adopted a similar approach by categorizing different types of student-generated prompts and applying statistical methods such as cluster analysis to interpret behavioral patterns, while also using qualitative techniques to explore student engagement types. Similarly, ref. [14] used a mixed-methods design, where they assessed the impact of student-generated prompts on ChatGPT performance through performance metrics and thematic analysis, offering deeper insights into prompting behaviors. These studies supported our decision to adopt a mixed research approach, as they show how combining both qualitative and quantitative techniques can provide richer, more comprehensive findings when studying how students interact with AI tools in educational settings.

2.2. Context and Participants

The investigation was conducted at a University of Technology in South Africa, within the Diploma in Software Development programme. All students registered for the diploma constitute the study's target population. For practical and curricular relevance, we narrowed the accessible population to those taking the four core software-development modules that span the three programme years. All the students in the study had prior knowledge of LLM tools and had used them in similar tasks. Using a census approach, we invited every student enrolled in those modules to obtain complete responses from 140 out of 160 students across the three cohorts. This strategy maximised coverage while preserving validity, because the data was collected during scheduled laboratory sessions that all students were already required to attend. The multi-year sample provides heterogeneity in prior programming experience and exposure to AI-assisted coding tools, thereby strengthening the transferability of findings. No demographic identifiers beyond year-of-study were collected.

2.3. Task Design and Data Collection Procedure

Data generation took place during laboratory sessions scheduled in each module's normal timetable. The session was not strictly supervised so as to give the students a relaxed atmosphere to complete the tasks. At the start of the session, lecturers issued a brief orientation covering study goals and the requirement that students operate under their own LLM credentials. All 140 participants signed into their personal ChatGPT, Gemini, or Claude accounts (whichever they had already used for coursework). This ensured that every prompt could be traced back to an individual session record in the LLM provider's history log. Each student then tackled at least three authentic software-development tasks, selected by the course lecturers to mirror weekly coursework. Students were free to query the LLM as often as needed; no time limits were imposed on individual tasks within the overall session window. All prompts and immediate LLM responses were later recorded and exported to CSV for analysis.

2.4. Data Preparation

Raw CSV files were first de-identified (replacement of e-mail addresses and platform IDs with random hashes) and merged into a single corpus ($N = 842$ prompts). System messages, greeting phrases and automatically appended code comments were removed with a custom Python 3.12.0 script. Each prompt was then sentence-segmented to enable coding of multi-question requests at their most granular meaning unit. The cleaned dataset, together with a lookup table linking prompts to the corresponding task, was produced for further analysis.

2.5. Data Analysis

The data analysis process followed a mixed-methods approach, combining descriptive quantitative techniques with qualitative thematic coding. To ensure the validity of our results, we used the typical workflow proposed in [15]. The aim was to identify patterns in prompting behaviour and interpret how these patterns reflected student learning strategies. Quantitatively, all 842 prompts were first categorized based on surface features such as prompt length, frequency per student, and LLM used. Descriptive statistics were calculated to determine the relative prevalence of different prompting styles. This included computing the percentage distribution of prompt types, which informed the dominant strategies reported in Section 3. Prompts were clustered into overarching categories representing prompting strategies. Each prompt was reviewed independently by two researchers to assign initial codes reflecting the prompt's purpose and cognitive demand. Discrepancies

were discussed and resolved through consensus. Inter-rater reliability was quantified using Cohen's κ , which yielded a value of 0.78, indicating substantial agreement between the two coders. This measure of reliability, proposed by [16] has been used and adopted in numerous studies [17,18].

To enhance analytical rigor, the researchers employed constant comparison methods to refine categories and ensure mutual exclusivity among the strategies. Bloom's Taxonomy [19] was used as an interpretive lens to assess the cognitive complexity of each prompt category, while elements of constructivist learning theory were applied to evaluate the extent of student engagement and conceptual growth. This combination of quantitative prevalence analysis and qualitative interpretive coding allowed for a better understanding of what students asked and how and why they interacted with LLMs during programming tasks.

3. Results

This study investigated the prompting strategies used by Information Technology students when working with LLMs to complete programming-related tasks. Students were asked to complete tasks across core programming modules as part of their coursework. While the task content varied, the prompts provided valuable insights into how students engaged with the AI tool. From this, we identified five dominant prompting strategies that influenced student learning, as shown in Figure 1.

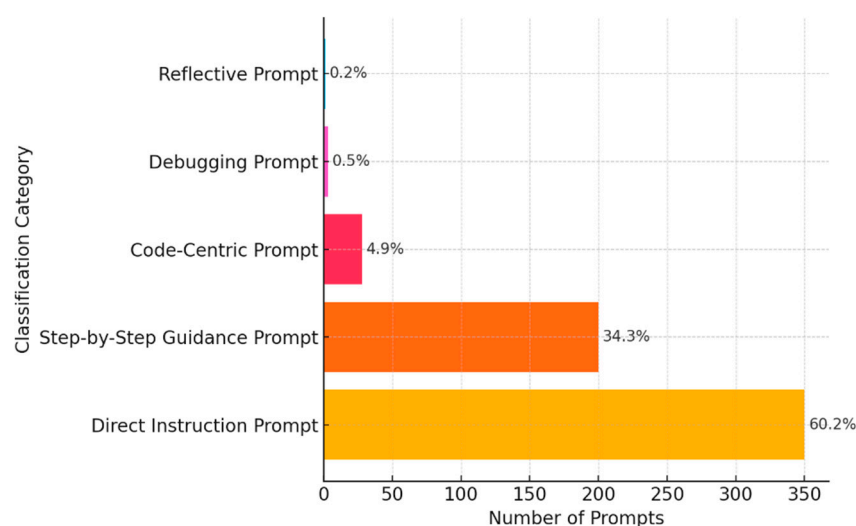


Figure 1. Distribution of Prompting Strategies Used by Students.

3.1. Direct Instruction Prompts

This was the most frequently used strategy, accounting for approximately 60.2% of all student prompts. Many students used prompts that asked LLM tools to define concepts or explain programming ideas. For instance, one student asked, "What is supervised learning and how does it work?", while another typed, "Explain the difference between classification and regression in machine learning". This kind of prompting is similar to what students might do when reading a textbook or asking a direct question in class.

Across the dataset, similar patterns appeared in prompts related to databases and programming tools. Examples included, "What is a relational database?" or "How do I connect to SQL in Visual Studio?" These types of prompts suggest that students were primarily aiming to understand key terms or foundational ideas before attempting the actual task. This strategy proved very popular with students. Most responses from the LLM were simple and easy to follow, which helped students get started. However, in many

cases, students did not ask follow-up questions or attempt to apply what they had learned unless the task explicitly required it. This limited their opportunity to explore the topic more deeply.

Therefore, while this type of prompt helped students obtain quick and relevant information, relying on it alone may not support deep or independent learning—especially when students do not push themselves to explore further. From a Bloom's Taxonomy perspective, these prompts reflect lower-order thinking skills such as "remembering" and "understanding". There was limited evidence of constructivist learning, as students relied on the tool for direct answers rather than building knowledge through active exploration. Similarly, there were few signs of self-directed learning, since students seldom followed up or reflected beyond the initial response.

3.2. Step by Step Guidance Prompts

This strategy made up 34.3% of the student prompts and reflected a more structured and deliberate approach. Some students broke their task into clear steps. One example involved a student who broke the task into clear steps, first asking what supervised learning is, then identifying beginner-friendly algorithms, and finally requesting a simple implementation in Python. A similar pattern continued in prompts such as, "How do I connect to a database in C#?", followed by "How do I insert data using a button click?", and later, "How do I display the inserted data in a DataGridView?" These prompts showed that the student was progressing from understanding the tools, to implementing functionality, and then verifying output all in a logical order.

The use of step-by-step guidance prompts indicates that students were not merely seeking final answers but were instead using ChatGPT to support a structured learning process. By breaking down complex programming problems into smaller, manageable steps—such as starting with a conceptual query ("What is supervised learning?") and progressing toward practical implementation ("Show me a minimal Python example")—students were able to guide the LLM interaction in a way that mirrored instructional scaffolding. This sequencing allowed them to build understanding incrementally, reducing cognitive overload and making it easier to process and apply new information. The prompt structure also appeared to encourage deeper engagement, as students refined their understanding by comparing outputs, testing solutions, and adjusting their queries accordingly. In this way, the LLM functioned as a responsive learning tool, offering just-in-time feedback that supported students in moving from basic comprehension to application and analysis. While this approach aligns well with self-regulated learning and constructivist principles, it also highlights a potential risk: students may become overly reliant on predefined prompt structures, which could hinder their ability to tackle more complex, less familiar problems independently. For educators, this underscores the importance of teaching students not only how to use AI tools effectively but also how to critically evaluate and adapt their own learning strategies over time.

3.3. Code Centric Prompts

This prompt style accounted for 4.9% of the data and was characterized by a strong focus on code generation. Some students jumped straight into implementation. Prompts included "Generate a classification model in Python using scikit-learn" and "Build a supervised learning model with Python code." Others submitted prompts like "Give me the full code to connect C# to SQL and add data" and "Build a CRUD app in Windows Forms." These students typically pasted the code into their development environment without much explanation or reflection.

The frequent use of direct code-generation prompts—such as “Write a Python function for. . .” or “Generate code to solve. . .”—suggests that some students were primarily focused on completing tasks rather than understanding the underlying principles. This pattern, especially common among novice programmers, points to a surface approach to learning, where the primary goal is task performance rather than conceptual mastery. Within the framework of Bloom’s Taxonomy, this strategy remains largely at the “remembering” and “applying” levels, with limited evidence of progression to “analyzing” or “evaluating”, which require deeper engagement with logic, structure, and reasoning. Students often accepted and submitted the first code output generated, rarely modifying or testing alternative solutions—an indication of minimal reflection or critical assessment. From a constructivist perspective, this approach underutilizes the learner’s active role in meaning-making and problem-solving. Constructivist learning emphasizes the construction of knowledge through experience, interaction, and reflection; however, these direct prompts suggest a transactional model of learning, where the LLM is treated as a source of ready-made solutions rather than a tool for exploration or conceptual growth. While these prompts can be helpful in time-constrained scenarios, they risk undermining the development of higher-order thinking skills and independent problem-solving capabilities. Thus, educators should encourage students to move beyond passive code consumption and engage in iterative processes—debugging, evaluating alternatives, and justifying choices—to support deeper learning and knowledge transfer.

3.4. Reflective Prompts

Only 0.2% of prompts were reflective, showing curiosity and a desire to understand reasoning or compare alternatives. Some students submitted prompts like, “Why is supervised learning better than unsupervised in some cases?” and “What are the pros and cons of using decision trees?” Others asked, “What are the different ways to connect a database in C#?” or “Which database type is best for a small inventory system?”. The use of troubleshooting prompts—such as requests for help with specific error messages or unexpected output—demonstrates a more reflective and diagnostic engagement with the LLM. These prompts typically occurred after students had already attempted a solution, indicating a shift from code generation to problem analysis. Within Bloom’s Taxonomy, this behaviour reflects movement into the “analyzing” and “evaluating” stages, where learners are required to identify the causes of problems, interpret system feedback, and make informed decisions about revisions. Unlike direct code-generation prompts, which often bypass critical reasoning, troubleshooting prompts necessitate a deeper cognitive investment. From a constructivist learning perspective, this interaction embodies the principle that knowledge is actively constructed through experience, reflection, and iterative problem-solving. When students used ChatGPT to examine errors and explore alternative explanations, they engaged in an authentic learning process that mirrors expert behaviour.

However, it is noteworthy—and concerning—that only a small percentage of prompts fell into this category. Given the nature of software development as a discipline grounded in debugging, problem decomposition, and iterative refinement, this limited use of troubleshooting strategies suggests a gap in how students are approaching computational thinking. The low frequency of these higher-order prompts raises pedagogical questions about students’ readiness to engage with the core practices of the discipline. Furthermore, the effectiveness of such prompts was highly dependent on how specifically and contextually they were framed. Vague queries often returned unhelpful responses, whereas well-structured prompts—accompanied by relevant code snippets or error logs—enabled more meaningful feedback. This indicates that while LLMs can support higher-order thinking, their value is maximized when students approach them as tools for hypothesis testing

and refinement rather than as simple troubleshooting shortcuts. Encouraging learners to articulate their debugging rationale not only enhances metacognitive awareness but also promotes the development of transferable skills essential for independent programming beyond AI-assisted contexts.

3.5. Debugging Prompts

Debugging accounted for 0.5% of the prompts, where students used the LLM to identify and resolve errors in their code. Examples included prompts such as, “Why am I getting a value error in this Python model?” and “How do I fix this scikit-learn import error?” Others asked, “Why won’t my SQL query insert data?” or “How do I fix a null exception in my C# form?”. This strategy reflected active problem-solving and trial-and-error learning. While it wasn’t always clear whether students fully understood the fixes, the prompts showed persistence and a willingness to engage with the debugging process.

Prompts that asked for explanations—such as “Explain what this code does” or “Why is this algorithm better?”—reflected a more conceptual engagement with the LLM and demonstrated an intent to understand rather than simply produce or fix code. These prompts align with the “understanding”, “analyzing”, and in some cases, “evaluating” levels of Bloom’s Taxonomy, as they require the learner to interpret, compare, and make judgments about code logic and design choices. From a constructivist learning perspective, such prompts indicate a shift from procedural knowledge to conceptual understanding. When students actively sought to clarify how or why a certain approach worked, they were engaging in reflective learning—constructing meaning through dialogue and integrating new information with prior knowledge. These interactions are pedagogically valuable, as they promote not only comprehension but also critical thinking and abstraction, which are essential for long-term skill development in software engineering.

However, similar to troubleshooting prompts, explanation-seeking prompts were relatively rare. This limited use is particularly concerning in the context of software development, a discipline that depends heavily on understanding how and why code behaves as it does. The low frequency of these conceptual prompts may signal a tendency among students to prioritise task completion over deep learning—potentially reinforced by the immediate utility and responsiveness of LLMs. Furthermore, the quality of these explanatory prompts varied significantly. Students who provided context—such as including code snippets or posing comparisons between algorithms—received richer, more accurate explanations. In contrast, vague or overly general questions yielded less informative responses. This suggests that while LLMs have the capacity to foster conceptual engagement, their effectiveness is mediated by the student’s ability to formulate thoughtful, specific queries. Supporting students in asking better questions—ones that elicit deeper understanding—should therefore be a key instructional goal. Doing so not only enhances metacognitive awareness but also cultivates the analytical mindset required for professional software development.

4. Discussion

The present study set out to determine whether students’ interactions with a LLM demonstrably supported learning in an undergraduate software-development context. The prompt log tells a consistent story: most exchanges sat at Bloom’s “remembering” and “applying” levels, with students requesting ready-made code or syntax explanations and seldom progressing to the “analyzing”, “evaluating”, or “creating” stages that typify expert programming practice [20]. Less than 12% of all prompts fell into the higher-order categories of troubleshooting or reflective explanation, mirroring patterns of surface engagement reported in similar AI-assisted coding studies [21]. In constructivist terms, learners treated

the model as an answer dispenser rather than a partner in knowledge construction, thereby bypassing the productive struggle that underpins conceptual change [22,23].

This limited cognitive reach is unsurprising when one considers the design of the prompts themselves. Straightforward requests such as “Write a Python function that sorts a list” require little cognitive elaboration; the LLM supplies a working snippet, and the task appears complete. Cognitive-load theory predicts that such minimal-guidance tasks reduce germane load—learning-relevant effort—while preserving extraneous load, because students do not need to organise or integrate new information [24,25]. The result, as our qualitative inspection confirmed, is code that “works” but a mental model that remains opaque. Without deliberate opportunities to interrogate, debug, or adapt that output, students accumulate what [26] call surface knowledge—rapidly gained, rapidly lost.

From a constructivist perspective, the scarcity of explanation-seeking and troubleshooting prompts is particularly troubling. Knowledge in software engineering is built through cycles of hypothesis, test, and revision [27]. When only a small minority of learners ask “Why is this algorithm preferable?” or “Explain the cause of this stack-trace”, opportunities to connect new code to prior mental schemata are forfeited. Consequently, the interaction pattern we observed cannot be said to exhibit learning in the strong sense of conceptual growth or transferable problem-solving skill [28]. We therefore make the following two recommendations to lecturers:

4.1. Re-Design Assessment Questions

Instructors can push students beyond procedural recall by structuring tasks that require higher-order engagement. Questions that ask students to critique alternative implementations, interpret error logs, or refactor inefficient code oblige them to operate at Bloom’s “analyzing” and “evaluating” levels. Empirical work in computer-science education shows that when prompts embed such metacognitive triggers, learners spend more time reasoning and produce higher-quality solutions [29]. Rubrics should therefore allocate explicit marks for explanation, justification, and iteration, not merely for a runnable script.

4.2. Teach Prompt Engineering as a Literacy

Prompting is itself a learnable skill [30]. Short workshops can demonstrate how to (a) provide sufficient context, (b) decompose complex problems, and (c) request stepwise reasoning or code annotations. When students use targeted prompts—e.g., “Given this traceback, describe two possible causes and suggest fixes with time-complexity estimates”—the LLM supplies feedback that naturally scaffolds inquiry, aligning with the fading-scaffold model of cognitive apprenticeship [31]. Early evidence indicates that explicit instruction in prompt design increases the proportion of explanation-seeking interactions by up to 30% and correlates with gains on post-test debugging tasks [32].

Together, these interventions reposition the LLM from an automated code generator to a catalytic tutor—one that supports learners in constructing, evaluating, and extending their own understanding. Future research should test the long-term impact of prompt-literacy curricula and higher-order assessment designs on students’ ability to transfer debugging and design skills to unfamiliar, LLM-free environments.

5. Conclusions

By exploring how IT students prompted LLMs during programming tasks, this study uncovered five core strategies that shaped their learning experiences. These ranged from seeking simple explanations to building complex solutions through structured and iterative dialogue. Each approach offered insight into how students used AI tools to support or sometimes short-circuit their understanding.

The analysis showed that while all five strategies were used by students, they varied in depth and impact. Direct Instruction helped with quick concept recall but rarely went beyond surface-level understanding. Step-by-Step Guidance encouraged deeper thinking and gradual skill-building. Code-Centric prompts offered speed but carried the risk of passive learning. Debugging Prompts highlighted problem-solving and resilience, while Reflective Prompts reflected critical thinking and evaluation.

What becomes clear is that prompting is not just a technical skill it's a learning habit. Students who approached LLMs tools with purpose, reflection, and a desire to understand are likely to get more out of the experience. These insights suggest that teaching students how to prompt more effectively through guided practice, modeling, or classroom discussions could strengthen their learning outcomes. In short, the way students engage with AI matters. By fostering better prompting habits, educators can help students become more independent, reflective, and capable learners in an AI-augmented educational environment.

Author Contributions: Conceptualization, T.N.M. and M.B.M.; methodology, B.B.H.; software, N.Z.N.; validation, J.M. and M.B.M.; formal analysis, T.N.M. and M.B.M.; investigation, B.B.H.; resources, J.M.; data curation, N.Z.N.; writing—original draft preparation, J.M. and B.B.H.; writing—review and editing, T.N.M. and J.M.; visualization, B.B.H. and M.B.M.; supervision, M.B.M.; project administration, M.B.M.; funding acquisition, M.B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Research Directorate at Mangosuthu University of Technology. The APC was funded by the Research Directorate at Mangosuthu University.

Institutional Review Board Statement: The study was approved by the Ethics Committee of Mangosuthu University of Technology (Ethical Clearance Number: RD1/15/2023, approved on 15 January 2023).

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study. The consent form for participation was distributed to all participants and signed.

Data Availability Statement: Data are not publicly available as they are protected under the ethics guidelines.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Keywords Everywhere. LLM Usage Stats: Key Insights for 2025. Available online: <https://keywordseverywhere.com/blog/llm-usage-stats/> (accessed on 25 June 2025).
2. Paustian, T.; Slinger, B. Students are using large language models and AI detectors can often detect their use. *Front. Educ.* **2024**, *9*, 1374889. [CrossRef]
3. Mnguni, N.M.; Nkomo, N.; Maguraushe, K.; Mutanga, M.B. An Experimental Study of the Efficacy of Prompting Strategies in Guiding ChatGPT for A Computer Programming Task. *J. Inf. Syst. Inform.* **2024**, *6*, 1346–1359. [CrossRef]
4. Hao, Z.; Jiang, J.; Yu, J.; Liu, Z.; Zhang, Y. Student engagement in collaborative learning with AI agents in an LLM-empowered learning environment: A cluster analysis. *arXiv* **2025**, arXiv:2503.01694. [CrossRef]
5. Azoulay, R.; Hirst, T.; Reches, S. Large Language Models in Computer Science Classrooms: Ethical Challenges and Strategic Solutions. *Appl. Sci.* **2025**, *15*, 1793. [CrossRef]
6. Felipe-Redondo, A.M.; Del Carmen-Morales, Y.A.; Del Carmen Morales, H.; Nuñez-Cárdenas, F.d.J. ChatGPT: Impact on programming learning among students at the Technological University of Huasteca Hidalguense. *Rev. Didáctica Práctica* **2024**, *8*, e3819110. [CrossRef]
7. Mutanga, B.M.; Lecheko, M.; Revesai, Z. Navigating the Grey Area: Students' Ethical Dilemmas in Using AI Tools for Coding Assignments. *IJIE Indones. J. Inform. Educ.* **2024**, *8*, 15–24. [CrossRef]
8. Mutanga, M.B.; Jugoo, V.; Adefemi, K.O. Lecturers' Perceptions on the Integration of Artificial Intelligence Tools into Teaching Practice. *Trends High. Educ.* **2024**, *3*, 1121–1133. [CrossRef]

9. Pudasaini, S.; Miralles-Pechuán, L.; Lillis, D.; Salvador, M.L. Survey on Plagiarism Detection in Large Language Models: The Impact of ChatGPT and Gemini on Academic Integrity. *arXiv* **2024**, arXiv:2407.13105. [CrossRef]
10. Yan, Y.-M.; Chen, C.-Q.; Hu, Y.-B.; Ye, X.-D. LLM-based collaborative programming: Impact on students' computational thinking and self-efficacy. *Humanit. Soc. Sci. Commun.* **2025**, *12*, 149. [CrossRef]
11. Jošt, G.; Taneski, V.; Karakatič, S. The Impact of Large Language Models on Programming Education and Student Learning Outcomes. *Appl. Sci.* **2024**, *14*, 4115. [CrossRef]
12. Perkel, S. College Students Are Using AI, But They're Conflicted About It. Business Insider. Available online: https://www.businessinsider.com/how-college-students-use-ai-homework-study-2025-3?utm_source=chatgpt.com (accessed on 28 March 2025).
13. Livingstone, V. I Quit Teaching Because of ChatGPT. Time. Available online: <https://time.com/7026050/chatgpt-quit-teaching-ai-essay/> (accessed on 28 March 2025).
14. Sawalha, G.; Taj, I.; Shoufan, A. Analyzing student prompts and their effect on ChatGPT's performance. *Cogent Educ.* **2024**, *11*, 2397200. [CrossRef]
15. Braun, V.; Clarke, V. Using thematic analysis in psychology. *Qual. Res. Psychol.* **2006**, *3*, 77–101. [CrossRef]
16. Landis, J.R.; Koch, G.G. An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers. *Biometrics* **1977**, *33*, 363–374. [CrossRef]
17. Warrens, M.J. Five Ways to Look at Cohen's Kappa. *J. Psychol. Psychother.* **2015**, *5*, 197. [CrossRef]
18. Schober, P.; Mascha, E.J.; Vetter, T.R. Statistics from A (Agreement) to Z (z Score): A Guide to Interpreting Common Measures of Association, Agreement, Diagnostic Accuracy, Effect Size, Heterogeneity, and Reliability in Medical Research. *Anesth. Analg.* **2021**, *133*, 1633–1641. [CrossRef]
19. Krathwohl, D.R. A revision of bloom's taxonomy: An overview. *Theory Into Pract.* **2002**, *41*, 212–218. [CrossRef]
20. Anderson, L.W.; Krathwohl, D.R. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives: Complete Edition*; Addison Wesley Longman, Inc.: Saddle River, NJ, USA, 2001.
21. Cui, J.; Zhao, Y.; Yu, C.; Huang, J.; Wu, Y.; Zhao, Y. Code Comprehension: Review and Large Language Models Exploration. In Proceedings of the 2024 4th IEEE International Conference on Software Engineering and Artificial Intelligence, SEAI 2024, Xiamen, China, 21–23 June 2024; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2024; pp. 183–187. [CrossRef]
22. Vygotsky, L.S. *Mind in Society: The Development of Higher Psychological Processes*; Harvard University Press: Cambridge, MA, USA, 1978; Volume 86.
23. Hmelo-Silver, C.E. Problem-Based Learning: What and How Do Students Learn? *Educ. Psychol. Rev.* **2004**, *16*, 235–266. [CrossRef]
24. Sweller, J. Cognitive Load During Problem Solving: Effects on Learning. *Cogn. Sci.* **1988**, *12*, 257–285. [CrossRef]
25. Kirschner, P.A.; Sweller, J.; Clark, R.E. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educ. Psychol.* **2006**, *41*, 75–86. [CrossRef]
26. Hattie, J.A.C.; Donoghue, G.M. Learning strategies: A synthesis and conceptual model. *npj Sci. Learn.* **2016**, *1*, 16013. [CrossRef]
27. Mcloughlin, C.; Lee, M.J.W. Personalised and self regulated learning in the Web 2.0 era: International exemplars of innovative pedagogy using social software. *Australas. J. Educ. Technol.* **2010**, *26*, 28–43. [CrossRef]
28. Marton, F. Approaches to Learning. *Exp. Learn.* **1997**, 39–58. Available online: <https://cir.nii.ac.jp/crid/1570291224332455296.bib?lang=en> (accessed on 28 May 2025).
29. Burden, H.; Steghöfer, J.-P. Teaching and fostering reflection in software engineering project courses. In *Agile and Lean Concepts for Teaching and Learning: Bringing Methodologies from Industry to the Classroom*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 231–262.
30. Haugsbaken, H.; Hagelia, M. A New AI Literacy For The Algorithmic Age: Prompt Engineering Or Educational Promptization? In Proceedings of the 2024 4th International Conference on Applied Artificial Intelligence, ICAPAI 2024, Halden, Norway, 16 April 2024; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2024. [CrossRef]
31. Collins, A.; Brown, J.S.; Holum, A. Cognitive Apprenticeship: Making Thinking Visible. *Am. Educ.* **1991**, *15*, 6–11.
32. Wang, T.; Zhou, N.; Chen, Z. Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering for Python Code Generation. *arXiv* **2024**, arXiv:2407.05437. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.