New Perspectives in Science Education
International Conference
NEW PERSPECTIVES in SCIENCE EDUCATION

# From Prompt to Pedagogy: Designing Effective LLM-based Programming Assistants Using Codeforces and HumanEval Datasets

**Rodolfo Bojorque[1], Fernando Moscoso[1]**

Universidad Politécnica Salesiana, Ecuador[1]

**Abstract**

*The emergence of Large Language Models such as ChatGPT and Gemini is redefining how students engage with programming education in higher education contexts. These models can generate explanations, examples, and personalized feedback, potentially transforming learning dynamics in computer science courses. However, their pedagogical integration remains underexplored, particularly regarding the reliability of automated feedback, students' critical engagement, and the risks of cognitive passivity or illusion of learning. This study investigates the pedagogical effectiveness of LLM-generated feedback by conducting empirical experiments using open datasets such as Codeforces and HumanEval, to simulate authentic student–AI interactions in coding tasks. Through a mixed-method design, the research evaluates feedback quality based on linguistic accuracy, conceptual relevance, and instructional value, employing both computational metrics (BLEU, CodeBLEU) and human assessments. Results reveal that carefully engineered prompts substantially enhance the coherence and instructional alignment of AI-generated feedback, promoting deeper conceptual understanding and learner autonomy. Yet, challenges persist in overreliance and the limited capacity of current models to foster metacognitive reflection. The study contributes to the growing discourse on the responsible and evidence-based integration of generative AI in higher education by offering design principles for effective feedback systems and proposing a pedagogical framework to balance automation with human guidance. These findings invite educators and institutions to rethink the boundaries between instruction, assistance, and assessment in the age of generative artificial intelligence.*

**Keywords:** *Large Language Models in Education; Programming Education; Prompt Engineering; Pedagogical Scaffolding.*

## 1. Introduction

Learning to program remains one of the most persistent pedagogical challenges in higher education computer science. Introductory and intermediate programming courses frequently report high failure and dropout rates, often attributed to students' difficulties in developing algorithmic thinking, limited opportunities for individualized feedback, and low confidence during problem-solving activities [1,2]. Programming education requires sustained practice supported by timely, precise, and pedagogically meaningful guidance—conditions that are difficult to guarantee in large classes with constrained instructional resources.

In recent years, the emergence of large language models (LLMs), such as ChatGPT, GPT-4, Claude, and LLaMA, has begun to transform how students interact with programming knowledge. These models demonstrate strong capabilities in generating, explaining, debugging, and translating code across multiple programming languages [3,4]. As a result, LLMs are increasingly adopted by students as informal programming assistants. However, their effective integration into educational contexts remains insufficiently understood, particularly regarding the pedagogical quality of the feedback they provide, students' critical engagement with AI-generated explanations, and the risk of cognitive passivity or illusion of understanding.

A key factor influencing the educational effectiveness of LLMs is prompt design. Prompt engineering—the structured formulation of input instructions—has been shown to substantially affect the correctness, coherence, and explanatory quality of model outputs [5]. In programming education, well-designed prompts may enable LLMs not only to generate functional code but also to provide step-by-

step reasoning, conceptual explanations, and instructional scaffolding aligned with students' levels of understanding. Despite its importance, empirical research examining prompt design from a pedagogical perspective remains limited, particularly in authentic programming scenarios.

To address this gap, this study investigates the pedagogical effectiveness of LLM-generated feedback through empirical experiments using two complementary open datasets: Codeforces and HumanEval. Codeforces is a widely used competitive programming platform that offers realistic, non-trivial algorithmic problems reflecting the challenges encountered by undergraduate computer science students beyond introductory courses [6]. HumanEval, by contrast, is a curated benchmark designed to evaluate code generation accuracy using standardized problem specifications and automated unit tests [3]. The combination of these datasets enables the analysis of LLM performance in both realistic and controlled educational contexts, allowing simultaneous evaluation of functional correctness and instructional quality.

Guided by this motivation, the study addresses the following research questions:

(i) How does prompt design influence the quality of code generated by LLMs in programming tasks?

(ii) Can the pedagogical value of LLM-generated feedback be systematically evaluated using combined computational and human-centered metrics?

(iii) What differences emerge when assessing LLM performance across benchmark-based (HumanEval) and real-world (Codeforces) programming contexts?

To answer these questions, we employ a mixed-method experimental design that applies different prompting strategies to both datasets and evaluates outputs along two dimensions: technical accuracy and pedagogical usefulness. Evaluation combines automated metrics with expert and student assessments to capture both functional performance and instructional value.

This work contributes to the growing discourse on the responsible integration of generative AI in higher education by offering empirical evidence on how prompt engineering shapes learning-relevant outcomes. By focusing on feedback quality rather than mere code correctness, the study proposes design-oriented insights for developing LLM-based programming assistants that balance automation with pedagogical intent and human guidance.

## 2. Related Works

### *2.1 Large Language Models in Programming Education*

Recent advances in large language models (LLMs) have substantially influenced how programming is learned and practiced in higher education. Models such as GPT-3 [7], Codex [3], and GPT-4 [4] have demonstrated strong capabilities in interpreting natural language instructions and generating syntactically correct and semantically meaningful code across multiple programming languages. These capabilities have positioned LLMs as promising tools for code autocompletion, debugging, explanation, and problem generation tasks.

Several studies have examined the educational use of LLMs in programming contexts. Zhang et al. [8] reported that students using ChatGPT as a coding assistant showed improved performance in practical assignments, although learning outcomes were strongly dependent on prompt clarity. Conversely, other studies caution that unsupervised or poorly guided use of LLMs may encourage surface-level engagement, code copying, and reduced conceptual understanding [9]. These findings highlight the need to investigate not only whether LLMs generate correct solutions, but also how they support learning processes.

### *2.2 Prompt Engineering and Pedagogical Considerations*

Prompt engineering refers to the structured design of input instructions aimed at improving the quality and relevance of LLM outputs. In educational settings, this practice extends beyond technical optimization to include pedagogical alignment, such as eliciting explanations, reasoning steps, and appropriate levels of detail based on learners' prior knowledge [10].

Sahoo et al. [5] proposed a taxonomy of prompt types, including direct, context-enhanced, and Socratic prompts, showing that prompt structure significantly influences output quality. In programming tasks, prompts combining functional specifications with explanatory guidance tend to produce more coherent and educationally useful responses. Reynolds and McDonell [11] further introduced prompt programming approaches that go beyond few-shot paradigms, enabling more systematic control over model behavior, although their application in educational contexts remains limited.

### *2.3 Educational Applications of LLMs with Code Datasets*

Benchmark datasets play a critical role in evaluating LLM performance in code generation tasks. HumanEval [3] is a widely used benchmark designed to assess functional correctness through standardized problem descriptions and automated unit tests, enabling reproducible quantitative comparisons. Codeforces, in contrast, provides real-world competitive programming problems characterized by higher algorithmic complexity and contextual variability. Banerjee et al. [12] demonstrated the usefulness of Codeforces-based evaluations for testing model robustness under realistic constraints.

Despite their complementary strengths, the pedagogical integration of these datasets remains underexplored. Kohen-Vacs et al. [13] investigated student interactions with AI-based programming tools in controlled environments, highlighting challenges related to error detection and trust. Qian et al. [14] showed that structured AI-generated feedback can enhance student motivation and engagement, reinforcing the importance of explanation quality alongside correctness.

### *2.4 Metrics for Evaluating LLMs in Code Generation*

LLM-generated code is commonly evaluated using a combination of functional and structural metrics. Pass@k [3] estimates the likelihood that at least one generated solution passes all test cases, while CodeBLEU [15] extends traditional BLEU by incorporating syntactic and semantic features specific to programming languages. Additional measures such as exact match accuracy and functional correctness provide complementary perspectives on model performance.

Beyond technical metrics, pedagogical evaluation has gained increasing attention. Liu et al. [16] proposed a Pedagogical Usefulness Score that integrates explanation clarity, conceptual depth, and language appropriateness, enabling assessment of instructional value rather than mere correctness.

### *2.5 Challenges, Limitations, and Research Gaps*

Despite their potential, LLMs face several challenges in educational contexts. Hallucinated or fabricated outputs remain a significant concern, particularly in programming tasks where subtle errors can undermine learning [17]. Furthermore, excessive reliance on code generation tools may lead to cognitive disengagement and skill erosion among students [18].

Dataset bias also poses limitations. While Codeforces emphasizes algorithmic problem-solving, it may not fully reflect industry-oriented software development practices. HumanEval offers greater standardization but is limited in scope and diversity. Although prior research has addressed individual aspects of LLM performance, important gaps remain: the lack of empirical comparisons between prompt types in educational settings, limited analysis of pedagogical feedback quality, and the absence of integrated frameworks combining benchmark and real-world datasets.

This study addresses these gaps by systematically evaluating prompt engineering strategies using both HumanEval and Codeforces, and by assessing LLM outputs through combined functional and pedagogical criteria grounded in educational theory [19].

## 3. Methodology

This study adopts a mixed-method experimental design to evaluate the pedagogical and technical effectiveness of large language model (LLM) outputs in programming education. The methodology integrates prompt engineering strategies, benchmark and real-world datasets, automated evaluation metrics, and human-centered assessments.

### *3.1 Datasets*

Two complementary datasets were used: HumanEval and Codeforces.

HumanEval [3] consists of 164 Python programming tasks defined by natural language specifications and validated through automated unit tests. It is widely used to assess functional correctness in code generation. To better support pedagogical analysis, each problem was annotated with metadata describing the main programming constructs involved (e.g., loops, recursion, data structures) and classified according to Bloom's taxonomy to distinguish between comprehension- and synthesis-oriented tasks.

Codeforces is a competitive programming platform containing real-world algorithmic problems of varying difficulty. A subset of 120 problems from Division 2 contests was selected and categorized by topic and difficulty level [6]. Since Codeforces problems do not provide standardized unit tests,

representative input–output cases were constructed based on problem statements and community discussions. Problem complexity was further characterized using cyclomatic complexity measures, enabling comparative analysis of generated solutions.
The combined use of HumanEval and Codeforces allows standardized evaluation while preserving ecological validity through realistic programming scenarios.

### *3.2 Prompting Strategies and Experimental Design*

LLM performance was evaluated using three prompt strategies:

1. Direct Prompt (DP): A concise task description requesting a solution.
2. Socratic Prompt (SP): Prompts incorporating reflective questions to elicit reasoning.
3. Scaffolded Prompt (ScP): Structured prompts guiding the model through step-by-step reasoning.

Each prompt type was applied to the same subset of 60 problems from each dataset. Experiments were conducted using the GPT-4 model via OpenAI's API (March 2024 version) with a temperature setting of 0.2. For each prompt–problem pair, five independent completions were generated, ensuring robustness against output variability.

### *3.3 Evaluation Criteria*

Evaluation was conducted along two complementary dimensions: technical correctness and pedagogical usefulness.
Technical Evaluation. Functional performance was assessed using Pass@1 and Pass@5 metrics [3], which estimate the probability that generated solutions pass all test cases. CodeBLEU [15] was used to measure structural similarity between generated and reference code, while cyclomatic complexity (McCabe Index) served as a proxy for code readability and structural clarity.
Pedagogical Evaluation. Instructional quality was assessed using a rubric adapted from Liu et al. [16], covering explanation clarity, conceptual depth, appropriateness of language, engagement potential, and misleading content. Outputs were rated on a 5-point Likert scale by two expert reviewers and two undergraduate students. Inter-rater reliability was calculated using Cohen's Kappa.
In addition, a group of 15 undergraduate computer science students reviewed a subset of outputs and provided feedback on perceived usefulness, clarity, and confidence. Responses were thematically coded and analyzed to complement quantitative findings.

## 4 Results

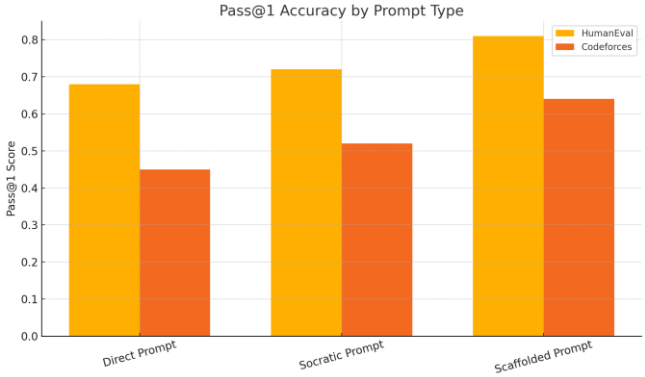
### *4.1 Overview*

This section presents the empirical results obtained from evaluating the outputs of GPT-4 under three prompting strategies (Direct Prompt (DP), Socratic Prompt (SP), and Scaffolded Prompt (ScP)) across two datasets: HumanEval and Codeforces. The analysis focuses on two dimensions: functional accuracy and pedagogical usefulness.

### *4.2 Pass@1 Performance*

Figure 1 summarizes the Pass@1 performance obtained for each prompting strategy across both datasets. In HumanEval and Codeforces alike, scaffolded prompts consistently outperformed direct and Socratic prompts, indicating that structured guidance substantially improves functional correctness. The observed trend is more pronounced in the benchmark dataset but remains consistent under the higher complexity of real-world programming tasks.

**Fig. 1:** Pass@1 accuracy by prompt type and dataset




### *4.3 Pedagogical Usefulness Ratings*

Figure 2 presents the average pedagogical usefulness scores assigned by reviewers. Across both datasets, outputs generated using scaffolded prompts achieved the highest ratings, followed by Socratic and direct prompting strategies. Scaffolded responses were more likely to include structured explanations, explicit reasoning steps, and justifications for algorithmic decisions, contributing to higher perceived instructional value.

### *4.4 Statistical Significance*

We conducted paired t-tests to compare prompting strategies:

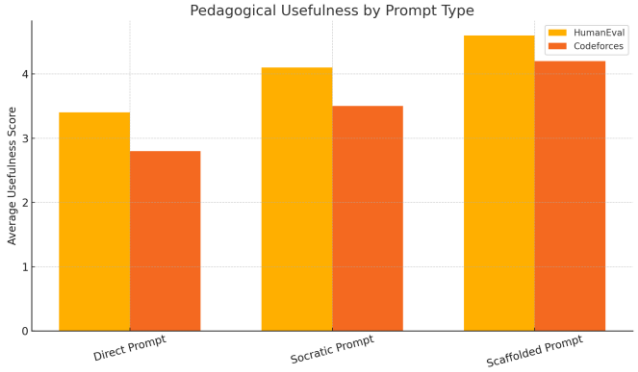
- **HumanEval** (DP vs. ScP): $p < 0.01$ for both Pass@1 and usefulness.
- **Codeforces** (SP vs. ScP): $p < 0.05$ for usefulness; marginal significance for accuracy ($p = 0.06$).

These differences confirm that scaffolded prompts lead to significantly better outcomes in both correctness and instructional value.

### *4.5 Additional Observations*

Additional analysis revealed that code generated through scaffolded prompts exhibited lower cyclomatic complexity and higher stylistic clarity, suggesting improved readability. Most participating students expressed a preference for scaffolded explanations due to their structured reasoning and transparency. In contrast, direct prompts more frequently produced syntactically correct but semantically flawed solutions, including incomplete logic or incorrect base cases.

**Fig. 2:** Pedagogical usefulness by prompt type and dataset




## 5 Discussion

### *5.1 Interpretation of Findings*

The results presented in Section 4 indicate that prompt structure plays a decisive role in shaping both the functional correctness and pedagogical usefulness of LLM-generated code. Across both

HumanEval and Codeforces datasets, scaffolded prompts consistently outperformed direct and Socratic prompts in terms of accuracy and instructional quality. This pattern aligns with established educational theories that emphasize guided instruction and cognitive scaffolding as key mechanisms for supporting learning in complex domains [19].

Scaffolded prompting appears to emulate expert instructional practices by decomposing programming problems into manageable steps and making reasoning processes explicit. This not only improves solution correctness but also enhances explanation clarity and conceptual depth. In contrast, direct prompts—while closer to authentic professional instructions—tended to produce concise but pedagogically limited outputs, occasionally containing subtle logical errors that may hinder conceptual understanding.

### *5.2 Pedagogical Implications*

The observed advantages of scaffolded and Socratic prompting have direct implications for the design of AI-assisted programming tools. Rather than positioning LLMs as passive code generators, these systems can function as instructional agents that model reasoning strategies, promote reflection, and support learner autonomy when guided by pedagogically informed prompt design.

From an instructional perspective, effective use of LLMs involves emphasizing reasoning processes over final answers, encouraging students to critically evaluate and refine AI-generated explanations, and treating prompt formulation as a pedagogical design decision rather than a purely technical task. Developing prompt literacy may therefore become a core competence in programming education, comparable to skills such as reading documentation or constructing test cases.

### *5.3 Limitations and Relation to Prior Work*

Several limitations should be considered when interpreting these findings. The experiments were conducted using a single LLM (GPT-4), and results may differ across other model architectures. In addition, only three prompt strategies were examined, and alternative approaches such as few-shot or chain-of-thought prompting may yield different outcomes. Although pedagogical usefulness was evaluated using both expert and student assessments, some degree of subjectivity is unavoidable despite acceptable inter-rater reliability.

This study extends prior work on LLMs in programming education by moving beyond correctness-focused evaluation [3] and incorporating explicit pedagogical criteria. While earlier studies reported positive effects of LLM-assisted programming [8], the present findings demonstrate that such benefits are highly sensitive to prompt design. By integrating real student feedback and comparing benchmark-based and real-world datasets, this work complements and expands existing simulation-based research.

## 6. Conclusions and Future Work

This study examined how prompt design influences the technical and pedagogical effectiveness of large language models in programming education. Through a controlled comparison of direct, Socratic, and scaffolded prompts across HumanEval and Codeforces datasets, the results show that scaffolded prompting consistently improves both functional correctness and instructional quality.

The main contributions of this work include: (i) an empirical comparison of prompting strategies in educational programming tasks; (ii) a dual-evaluation framework combining functional accuracy and pedagogical usefulness; and (iii) evidence-based design insights for developing LLM-based programming assistants that support learning rather than mere solution generation.

Despite these contributions, limitations remain regarding model diversity, prompt scope, and the absence of longitudinal classroom deployment. Future research should therefore explore adaptive prompting systems tailored to learner profiles, conduct in-classroom longitudinal studies to measure learning gains and retention, and investigate multimodal explanations that combine code with visual or conceptual representations. Ethical considerations such as transparency, academic integrity, and equity should also be foregrounded as LLMs become increasingly embedded in educational practice.

Overall, the findings suggest that prompt engineering is not simply a technical optimization strategy but a pedagogically meaningful design practice. When thoughtfully integrated, LLMs can function as scalable instructional partners that support reasoning, explanation, and conceptual understanding in programming education.

**REFERENCES**

[1] Watson, C. and Li, F. W. (2014). Failure rates in introductory programming revisited. Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, pages 39–44.
[2] Luxton-Reilly, A. (2016). Learning to program is easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, pages 284–289. ACM.
[3] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.
[4] OpenAI (2023). Gpt-4 technical report. Technical report, OpenAI. arXiv preprint arXiv:2303.08774.
[5] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927. Preprint.
[6] Ponomarenko, V. (2020). Codeforces: Competitive programming platform. ACM Inroads, 11(4):79–82
[7] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901.
[8] Zhang, Z., Dong, Z., Shi, Y., Matsuda, N., Price, T., and Xu, D. (2023). Students' perceptions and preferences of generative artificial intelligence feedback for programming. arXiv preprint arXiv:2312.11567. Preprint.
[9] Diyab, A., Frost, R. M., Fedoruk, B. D., and Diyab, A. (2025). Engineered prompts in chatgpt for educational assessment in software engineering and computer science. Education Sciences, 15(2).
[10] Zhou, Q., Chen, X., and Yin, Y. (2023). Socratic prompting in large language models for education. Proceedings of the 2023 International Conference on Learning Representations (ICLR).
[11] Reynolds, L. and McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. arXiv preprint arXiv:2102.07350.
[12] Banerjee, S., Patra, R., and Pradhan, D. (2022). Codeassist: A codeforces-based evaluation framework for educational code generation. In IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 1–10.
[13] Kohen-Vacs, D., Usher, M., and Jansen, M. (2025). Integrating generative ai into programming education: Student perceptions and the challenge of correcting ai errors. International Journal of Artificial Intelligence in Education.
[14] Qian, C., Liu, T., and Zhang, M. (2023). Generative ai for automated code feedback: An empirical study. ACM Transactions on Software Engineering and Methodology, 32(2).
[15] Ren, S., Yang, Z., Lin, B., and Sun, M. (2020). Codebleu: A metric for evaluating code generation. arXiv preprint arXiv:2009.10297.
[16] Liu, Y., He, S., and Ren, X. (2023). Evaluating pedagogical usefulness of llms: A human-ai collaboration study. pages 2304–2315.
[17] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., et al. (2023). Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12).
[18] Kumar, M., Singh, A., and Kaur, R. (2022). Student reliance on code-generation tools: A double-edged sword. Journal of Educational Technology & Society, 25(3):18–29.
[19] Chi, M. T. H., de Leeuw, N., Chiu, M.-H., and Lavancher, C. (1994). Eliciting self-explanations improves understanding. Cognitive Science, 18(3):439–477.