Academic Education Publishing House
Occupation and Professional Education Vol. 3 No. 9, 2026

A Study of the Teaching of Python Programming at Secondary Vocational Levels with a View to Generative Artificial Intelligence

Shangyi Cai[1] ,Yanna Feng[2*]
[1]College of Applied Science and Technology,Beijing Union University, Beijing, China
[2]Beijing Union University, Beijing, China
**Corresponding Author*

Abstract:Generative artificial intelligence may offer an innovative solution to some of the problems of secondary vocational Python teaching. The present article takes up the students' cognitive standpoint, looks at AI's role in instruction, and formulates a basic design principle—limited, moderate, or full—for the degree of AI involvement. Regarding BOPPPS, three defects in secondary vocational contexts are analyzed: pre-assessment not detecting difficulties, participatory learning without adequate task design, and post-assessment neglecting understanding. It suggests connecting pre-assessment with AI-centered diagnosis of learning, using task-oriented approaches for participatory learning, and adding a "code explanation" item to post-assessment. Then it outlines—on an exploratory basis—an overall scheme for pre-class, in-class, and post-class stages.

Keywords:Generative Artificial Intelligence; Secondary Vocational Training; Python Code; BOPPPS Model; Teaching Experimentation

1. Introduction

Python is becoming more central to secondary vocational courses involving computers, though teaching remains problematic[1]. Python's abstract notions do not quite fit with students' tendency toward image-based thinking. Skill levels vary widely among learners. Loops and recursion present substantial difficulty. Standard approaches—whether teacher-centered or not—are too time-consuming and fail to help much individually. Current writings give scant attention to practical details of AI operation[2]. The present paper deals specifically with vocational Python teaching. It gives an account of AI functions, of usage and of guidance, treats BOPPPS as an in-class model, and adapts it for secondary vocational settings.

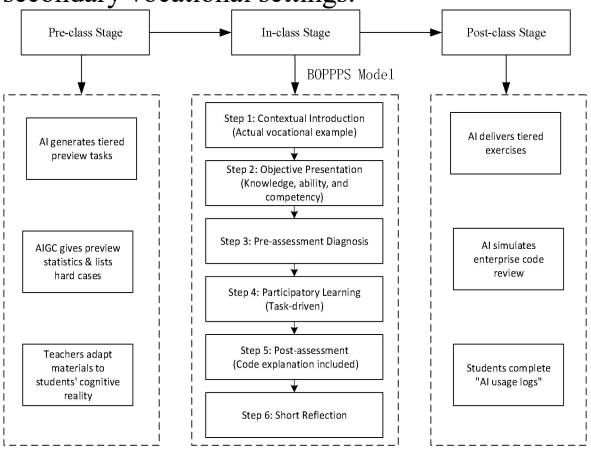


Figure 1:AI-Assisted Python Teaching Framework in Secondary Vocational Schools

2. Description of Instructional Role and of Boundaries for Generative AI in Connection with Secondary Vocational Python Courses

2.1.Three Teaching Roles of AI

AI is not to be considered as applicable at every teaching level. Intervention—minimal, moderate, or complete—must be decided separately for each stage[3]. For ideas, problems, or solutions, minimal intervention: students first work on understanding and design, then apply AI to check or combine, not omitting a cognitive step. Writing and optimizing code require moderate intervention: AI gives hints, not definite solutions; students find and fix errors. Use of resources, tiered exercises, or formatted code all make use of full intervention. Such an approach is grounded in an analysis of secondary vocational students' cognition. They normally apply AI immediately; they do not handle self-regulation or timing of AI well. It is therefore wise to control intervention carefully to ensure that ideas come about via genuine cognitive work.

2.2 On the Question of Boundaries for the Degree of AI Intervention

Limited intervention covers concept understanding, problem analysis, and solution design. For concept understanding, students first use their own experience, then apply AI as an aid, not a final definition, without treating given meanings as complete truths. Problem analysis starts with personal reflection or a simple estimate, using AI as a check, not a shortcut from vague facts to definite answers. Solution design involves choice, testing, and revision; afterwards, AI may evaluate advantages and drawbacks.

Moderate intervention covers code writing, debugging, and optimization. AI gives prompts or diagnosis, not fixed solutions; students fix errors. Full intervention covers all instructional material, tiers of work, and code standards.

This design considers students' cognitive features. They tend to adopt AI outputs without much discrimination of when to apply AI. Thus, intervention must be planned carefully so that basic ideas come about through cognitive processing.

3. Pre-class Stage: AI-Assisted Preview with Tiering and Learning

Teachers input course content and class facts into AIGC tools to generate tiered preview tasks. For loop statements: basic—syntax explanation or line-by-line work; intermediate—short imitation writing; advanced—loop changes. AIGC reports preview statistics and hard cases, helping teachers decide class emphasis. Since secondary vocational students have weak autonomous learning, preview should not rely wholly on self-discipline. AI reduces cognitive difficulty with tiered tasks—e.g., fill-in-the-blank instead of open-ended—avoiding early discouragement. Teachers also use AI to make plans, show cases, or compile errors. Selected material must be examined and adapted to students' cognitive reality[4].

4. In-Class Stage: An Attempt at AI-Assisted Teaching Using Improved BOPPPS

4.1 On the Advantages and on the Deficiencies of the BOPPPS Model in Secondary Vocational Use

Among the six BOPPPS steps—Bridge-in, Objective, Pre-assessment, Participatory Learning, Post-assessment, and Summary—one finds a full classroom loop. What is done is clearly laid out, so that AI intervention points may be easily located or defined[5].

There are three definite shortcomings of this model in secondary vocational settings. First, pre-assessment tends too readily to become formalistic. Students' wrong answers—whether from ignorance or from evasion—do not help to push forward an investigation or an education. Second, participatory learning lacks clear design. BOPPPS prescribes proportions of time but not of learning. Absence of tasks gives rise to wasteful arrangements such as "students read code while the teacher circulates." Third, post-assessment is not fully reliable in measuring understanding. Standard post-assessment deals with coding, not with explanations. With AI help, "AI can write but students cannot explain" is more likely to happen[6].

4.2 Improvement Design for the Shortcomings

Improvement 1: Link pre-assessment with AI learning diagnosis. Use a two-layer design: pre-class AI diagnosis plus in-class verification. Before class, AIGC records preview information. In class, the teacher comments on likely problems from AI; students may ask or raise hands. This addresses reluctance to reveal difficulties and makes pre-assessment a focus; no anonymous data needed.

Improvement 2: Make participatory learning task-driven. Activities: requirement analysis and solution design (limited AI); AI-aided coding (moderate AI); debugging and optimization (moderate AI); explanation and reflection (full AI). Steps, AI use points, and outputs must be clearly defined.

Improvement 3: Add code explanation to post-assessment. Students explain significant parts of written code and their logic[7]. If only code is tested, students can copy AI-produced code and pass without genuine knowledge. With explanation, logical comprehension becomes unavoidable.

4.3 Description of the Improved Six BOPPPS Steps

Step 1: A contextual introduction (about 5 minutes). An actual vocational example—the "Student Grade Management System"—gives the task. The students do an analysis of functions and sketch a structure diagram. AI is not applied; it refers to initial problem identification.

Teachers are encouraged to use "think first, ask later."

Step 2: Objective presentation (about 2 minutes). Knowledge, ability, and competency may be considered three levels. Basic objective is to understand and alter programs. Intermediate: apply AI in connection with modules. Advanced: design freely and assess AI-derived code. Such tiering gives a minimum standard for each level.

Step 3: Pre-assessment diagnosis (of roughly 5 minutes). The teacher uses AIGC to obtain a learning diagnosis, to list problematic knowledge items, and to ask for student confirmation (by hand or by short reply) of understanding. Anonymous handling is less stressful.

Step 4: Participatory learning (about 25 minutes). Groups program using the "Student Grade Management System." First, students study the problem and divide it into modules with restrictions on AI use. They then proceed to write code module by module; if stuck, they turn to AI for help in verification. AI gives structured hints: it names the error, identifies key facts, and proposes corrections—but not full solutions. At this stage the whole is debugged, and AI acts as an enterprise reviewer. Each group presents its work, giving details of AI-aided steps, suggestions taken or discarded, and reasons.

Step 5: Post-assessment (about 5 minutes). A short programming task and an explanation of code are included. The first is a test of code-writing ability; the second asks students to interpret or describe a code segment logically. The teacher chooses 2–3 people at random for oral explanation; others respond by writing. This helps to identify those who can execute code yet fail to grasp logic, and so to plan tiered tutoring.

Step 6: Short reflection (about 3 minutes). The teacher checks on the students' knowledge. Two or three of them are to state where they applied AI, what they asked, how AI was helpful, and whether they met any AI errors. Encountering AI errors is a useful way of seeing that AI need not be right.

5. Post-class Stage: AI Assistance with Consolidation, Expansion, and Personal Growth

AI is to be advised to make use of classroom data in connection with tiers. The basic tier deals with syntax and with correction of errors; the advanced tier has functions and error analysis; challenge calls for projects or for solution work. The basic tier must not be too hard, otherwise students quit.

AI tools now simulate real-life review of an enterprise's code, and treat uncommented functions, hard-coded values, or missing exceptions as engineering problems. Such simulation is useful and helps establish good practice. The rule is stated as "code should not only run, but also be readable and maintainable[8]." Students are to make a short "AI usage log," which gives context, asks questions, and lists AI errors found. Use of a table or of choices is fine; longer writing need not be done.

6. Problems and Reflections in Teaching Exploration

AI-generated content may not suit students' current levels. For example, dictionary comprehensions or lambda expressions are too hard for most vocational students. Teachers must simplify output. They can preset difficulty limits and, when needed, prompt AI to restrict syntax or keep code short[9].

Students vary in questioning ability. AI's individualized help depends on asking questions, but students may not know how. Instruction in prompts, guided questioning, or models is needed[10].

Over-dependence and cognitive offloading are harmful. If students hand all cognition to AI, it may help with syntax but not architectural reasoning. The most workable anti-dependence measure is code explanation. When students must explain logic, they are unlikely to copy code or bypass structural meaning.

The six BOPPPS steps may not all fit one class session. In 45 minutes, summary or post-assessment becomes too rushed. BOPPPS time allocations are references, not rules. Splitting task-oriented activities across two class periods is feasible.

7. Conclusion

Generative AI is worth considering in secondary vocational Python teaching. The present paper must not only list AI functions but also specify stages and levels of intervention. It treats BOPPPS as part of classroom organization and remedies three problems: pre-assessments too hard to implement, learning without real activity, and post-assessments failing to show comprehension. The central idea is that AI intervention aims at cognitive engagement—not

at mere technology, nor at arbitrary judgment[11]. The scheme outlined here still awaits practice and has room for further development along these lines. Later work might isolate, for each type of intervention, the role it plays—or fails to play—in knowledge growth and cognition.

References

[1]Stone, I. (2024) Investigating the Use of ChatGPT to Support the Learning of Python Programming Among Upper Secondary School Students: A Design-Based Research Study. In The United Kingdom and Ireland Computing Education Research (UKICER 2024), ACM, New York, NY, USA, 1 page.

[2]Margulieux, L.E., Prather, J., Reeves, B.N., Becker, B.A., Cetin Uzun, G., Loksa, D., Leinonen, J. and Denny, P. (2024) Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. In Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), ACM, New York, NY, USA, 7 pages.

[3]Fan, Y., Tang, L., Le, H., Shen, K., Tan, S., Zhao, Y., Shen, Y., Li, X. and Gašević, D. (2025) Beware of Metacognitive Laziness: Effects of Generative Artificial Intelligence on Learning Motivation, Processes, and Performance. British Journal of Educational Technology, 56, 489-530.

[4]Nathaniel, J., Oyelere, S.S., Suhonen, J. and Tedre, M. (2025) Investigating the Impact of Generative AI Integration on the Sustenance of Higher-Order Thinking Skills and Understanding of Programming Logic. Computers and Education: Artificial Intelligence, 9, 100460.

[5]Liu, J., Omar, S.Z., Wang, Y. and Xiang, Y. (2025) BOPPPS Model Implementation and Students' Performance: A Systematic Literature Review. Frontiers in Education, 9, 1467225.

[6]Kumar, A., Wongsirichot, T. and Nanthaamornphong, A. (2026) Generative AI in Computing Education: A Systematic Review and a Framework for Responsible Integration. Computers and Education: Artificial Intelligence, 11, 100677.

[7]Adeeb, E. and Muldner, K. (2025) How Do Novice Programmers Solve Code-Tracing Problems When ChatGPT Is Available? A Qualitative Analysis. In Proceedings of the 2025 ACM Conference on International Computing Education Research (ICER 2025), ACM, New York, NY, USA.

[8]Oliveira, E., Fu, M., Thongtanunam, P., Lopez-Pernas, S. and Saqr, M. (2026) AI-Assisted Code Review as a Scaffold for Code Quality and Self-Regulated Learning: An Experience Report. In 2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-SEET '26), ACM, New York, NY, USA, 11 pages.

[9]Zi, Y., Li, L., Guha, A., Anderson, C.J. and Feldman, M.Q. (2025) "I Would Have Written My Code Differently": Beginners Struggle to Understand LLM-Generated Code. In 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), ACM, New York, NY, USA, 10 pages.

[10]Sun, D., Boudouaia, A., Yang, J. and Xu, J. (2024) Investigating Students' Programming Behaviors, Interaction Qualities and Perceptions through Prompt-Based Learning in ChatGPT. Humanities and Social Sciences Communications, 11, 3991.

[11]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, 35, 3166-3184.