ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

Generative AI-Enabled Formative Assessment in a Higher Vocational Surveying Programming Course: Design and Initial Practice

Shuolin Meng

Department of Geomatics Engineering, Shijiazhuang Institute of Railway Technology, Shijiazhuang 050041, China

Abstract

Programming courses in higher vocational geomatics education ask students to combine Python syntax with surveying computation fundamentals, data-processing workflows, and engineering-quality standards. In many cases, assessment of traditional assignments is both delayed and oriented toward final results, which makes it hard to pinpoint issues such as algorithmic mistakes, code-quality shortcomings, and each learner's specific needs. This study proposes a generative AI–enabled formative assessment model for the Surveying Programming (Python) course. Using a platform that was developed in-house, the model integrates rule-based checks, test cases, domain-specific rubrics, large-language-model analysis, and teacher review to create a closed loop that supports task design, code submission, intelligent evaluation, personalized feedback, revision, learning analytics, and ongoing instructional improvement. The model was first piloted in the spring 2026 term in Engineering Surveying Class 2401 (n = 42). Data aggregated from five assignments were analyzed descriptively. The platform dashboard showed a 95% assignment completion rate, an overall average score of 84.6, and four students who required additional attention. Mean assignment scores rose from 78.1 on the first task to 88.9 on the fifth task. The most frequently recorded problems were file-reading exceptions (28%), non-standard variable naming (22%), and loop-boundary errors (18%). Overall, these results offer early evidence that the model is feasible for providing timely, domain-informed feedback and for turning programming-process data into actionable teaching information at the class level. Because this study involves only one class and relies on descriptive platform data without a control group, the findings should be viewed as implementation evidence rather than proof, in a causal sense, of improved learning effectiveness.

Keywords

Generative artificial intelligence; formative assessment; surveying programming; higher vocational education; code feedback; learning analytics.

1. Introduction

The integration of AI is increasingly transforming the landscape of education from a supportive digital tool to a central component of the educational landscape. Specifically, the "Artificial Intelligence + Education" Action Plan of China relates to AI-assisted learning and teaching, assessment, student digital profiles, and data-driven governance, all of which are explicitly promoted [1]. In parallel, the Education Powerhouse Development Plan (2024–2035) sets out to create a quality vocational education system that aligns with industrial transformation and boosts digital skills [2]. These policies, combined, create a supportive environment to explore and experiment with how generative AI can help with everyday teaching rather than with general-purpose question answering.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

This change means that formative assessment becomes even more crucial. Formative assessment is different from assessment in terms of verifying learning outcomes at the end of the learning process, because formative assessment is used to collect a variety of evidence throughout the learning process and is used to direct what students do and to inform what teachers do. The purpose of good feedback is to make the learner aware of the desired performance, to help him or her recognize where the current performance is lacking, and to determine what steps can be taken next within the learner's reach [5–7]. But in the context of programming education, this type of feedback is difficult to achieve. Each submitted program must be run by the teacher, error messages must be understood, the students' algorithmic thinking must be explored, intermediate calculations must be checked for correctness, and the teacher must explain how the program might be changed. If a class turns in numerous similar, but not exactly the same programs, feedback will probably be delayed or consist of a grade and a brief comment.

There are tools that can be automated to check if a program compiles or if the program produces the correct output. But many of these systems seem to be concerned only with correctness and unit-test performance, with much less concern about readability, maintainability, clear explanations, or domain reasoning, as suggested by systematic reviews [8, 9]. Things have changed with generative AI, however, as it understands code and can provide natural language feedback, including multi-level recommendations and questions [10-13]. However, LLMs may make mistakes, provide contradictory assessments, reveal full answers prematurely, or simply regurgitate biases. Therefore, it is necessary to provide constraints, deterministic checks, clear and transparent rubrics, and teacher supervision for educational applications [3, 11-13].

These problems are particularly noticeable in Surveying Programming (Python), a practical class for the higher vocational geomatics student. Students are not only required to write correct programming syntax, but also to apply the surveying formulas and workflows such as propagation of azimuth, calculation of closure-errors, allocation of corrections, elevation adjustments, calculation of coordinates, working with files and standardized reporting of the results. A program can execute without error but not satisfy the requirements for surveying logic or accuracy. A program can execute without error but not satisfy the requirements for surveying logic or accuracy. On the other hand, a student could have a good grasp of the underlying surveying principle but end up with a problem such as an incorrect path, converting data types, or wrong loop boundaries. That's why grading should be based on the code as well as the professional surveying process.

The studies conducted in the recent IJOSSER include explainable feedback through LLM, practical training with closed-loop feedback using AI, and reform of Python programming courses using AI [15–17]. Even so, there remains a practical disconnect between the source code, runtime data, knowledge surveying, repeated submissions, personalized guidance, class diagnosis, and teacher verification in the domain-specific formative assessment process. To address this, this study is designed and presented an AI-based formative assessment model that is integrated into a realistic surveying programming course.

How might generative AI be applied in conjunction with rules, test cases, domain rubrics, and teacher review to evaluate surveying programs? (2) How can code and process data translate into personalized feedback and class level learning analytics? (3) What are the trends in the early implementation data, and what are some of the considerations for broader adoption?

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

2. Conceptual Basis and Research Gap

2.1. Formative Assessment as a Feedback Loop

Formative assessment is described by Black and Wiliam as the use of evidence to change the way teaching and learning is carried out [4]. This perspective shifts the emphasis from the frequency of tests from the use of assessment information by teachers and students. Effective feedback, according to Hattie and Timperley, helps to answer three important questions: Where am I going? How am I doing? What do you want to do now? [5]. Nicol and Macfarlane-Dick also connect feedback with self-regulated learning, noting that students are active learners, have opportunities for reflection and can respond to feedback [6]. Shute also notes that feedback is most effective when it is specific, timely, encouraging, and task-oriented, not learner oriented [7].

In keeping with this, the current model does not see AI scoring as the end goal of assessment. Rather, it follows a closed loop learning process in which students submit their code, get clear and actionable feedback, edit their code, resubmit, and then show progress using the revised version as evidence. Teachers then combine all of the outcomes of these cycles to identify common issues and adjust for future teaching. This means that there are two formative roles that are supported on the platform: the micro-level role of providing individual student guidance, and the macro-level role of course and classroom diagnosis.

2.2. Automated and Generative Feedback in Programming Education

Traditional automated assessment usually depends on things like compilation, unit tests, output matching, static analysis, or comparing the student's work to a reference solution. These methods are reliable and repeatable for verifying syntax and output, yet they often can't clarify why a surveying formula was applied incorrectly or suggest how the program might be reorganized. Studies reviewing automated programming feedback have repeatedly pointed out the need for more detailed explanations and a wider evaluation of code quality [8, 9].

Large language models can add value alongside deterministic checks by analyzing code structure, guessing likely misunderstandings, and delivering feedback in language that's easier to understand. Nguyen and Allan showed a tiered formative-feedback method aimed at conceptual understanding, syntax, complexity, and suggested next steps [10]. More generally, research on generative AI in education highlights opportunities for personalization and fast support, while also stressing issues such as reliability, transparency, privacy, academic integrity, and the need for human supervision [11-14]. Altogether, these results indicate that effective educational design should not be purely manual or purely automated. Instead, a hybrid approach can combine deterministic methods for outcomes that can be verified, generative AI for explanation and guidance, and teacher review for the final decision and for exceptional cases.

2.3. Domain Specificity of Surveying Programming

Surveying programs are not the same as general introductory exercises since they must be correct according to domain-specific rules. They include, for example, angle normalization, the direction of azimuth propagation, acceptable closure errors, sign conventions, proportional correction, output formats used in engineering practice, units, and significant figures, as well as assumptions regarding the coordinate reference. This requires domain-specific evaluation to check intermediate values and key decision points, not only the outcome. This is why we use a course task bank, knowledge-point tags, reference calculations, and task rubrics in combination with the language model.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

3. Research Context and Method

3.1. Course Context and Participants

Surveying Programming (Python) is a course which is taught to students who are enrolled in higher vocational courses in surveying and mapping. The goal of the course is to enable students to read, calculate, organize, analyze, and present surveying data using Python. It covers Python basics, input/output with files, coordinate calculation, traverse and leveling data processing, calculation of curve elements, and analysis of deformation-monitoring data, among other applications. Students create multiple forms of supporting evidence for each task (e.g., source code, input files, outputs, run-time log, error messages, submission times, revision history, feedback responses).

This is the first time the initial practice described here was performed in the spring semester of 2026 with Engineering Surveying Class 2401. A class dashboard displayed consolidated data for 42 students and 5 class assignments. Class-level information was only reported in this study. This manuscript does not contain any individual student names, codes, or profiles.

3.2. Research Design

A design-based descriptive case study was used. First, the research team pinpointed the assessment issues in the course. They then designed the platform-supported feedback model, incorporated it into the course tasks, and finally analyzed the process and dashboard evidence that emerged. This is suitable for an early-stage teaching innovation where the aim is to record the design rationale, test operational feasibility and check the model's implementation in practice before embarking on larger comparative evaluations.

The evidence presented consisted of evidence of platform functions, course task requirements, scoring dimensions, aggregate submission and completion rates, evidence of knowledge point mastery, error-type distributions, and teacher observations derived from platform use, among other things. Descriptive statistics and instructional interpretation were used to analyze. As there was no control group, pretest, validated attitude scale, and no inferential statistical design, any changes in scores cannot be attributed to the AI intervention alone.

Table 1. Data elements used in the formative assessment process

Data element	Source	Typical content	Instructional use
Code data	Student Python files	Functions, variables, algorithms, comments, module calls	Assess programming implementation and surveying logic
Runtime data	Execution and test results	Status, outputs, precision, errors, runtime	Check correctness, stability, and engineering output
Process data	Submission and revision records	Time, attempts, returns, feedback viewing	Observe persistence, debugging, and revision behavior
Assessment data	AI analysis and teacher review	Dimension scores, reasons, adjustments, feedback text	Standardize criteria while retaining human judgment
Profile data	Accumulated task records	Mastery, error patterns, ability level, risk alerts	Support personalized tutoring and class diagnosis

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

3.3. Reliability, Privacy, and Human Oversight

It is designed according to a "teacher-led, AI-assisted" strategy. In practice, the tasks are developed by teachers, the professional calculation procedure is outlined, test cases or accepted standard results are set, the assessment criteria are specified, the teacher reviews the cases that are unusual, and the final evaluation is validated. High-stakes decisions are not made by the language model alone. The platform features aggregated reporting for research presentation, and individual learning profiles are stored as a tool to guide instruction and not as a ranking system.

4. Design of the AI-Enabled Formative Assessment Model

4.1. Closed-Loop Process

The model has six interrelated stages connecting assessment to action. The teacher first sets up the task, identifies the important knowledge points of surveying, and defines the evidence and criteria. The platform then collects the source code, the runtimes, error logs, and data from the submission process. Thirdly, it combines deterministic tests, subject-specific rules, language-model interpretation, and teacher review in a hybrid evaluation engine. Fourth, the feedback generated by the system is personalized and clearly indicates where the error has occurred, the most probable cause of the error, which calculation was affected, and what revision is recommended. Fifth, students revise the program and re-submit it. Finally, the system uses aggregated records for learning analytics and improvements to teaching, which then informs the design of the next round of task and rubric.

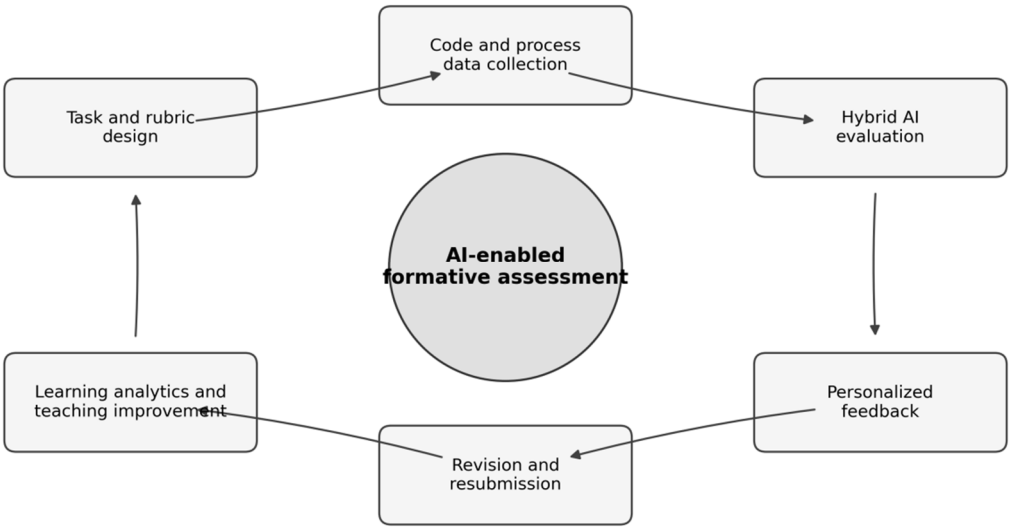

Figure 1. Closed-loop model of generative AI-enabled formative assessment

4.2. Platform Architecture

The self-developed platform has student, teacher, and management portals. The student portal allows students to see tasks, upload their code, submit the output from the code, see feedback, and resubmit if necessary. The teacher portal enables tasks to be released, rubrics to be set up, code reports to be viewed, AI scores to be checked, and class statistics and teaching-analysis reports to be generated. The management portal consolidates course, class, task and semester data, enabling the course-target attainment analysis and helping to improve the professional program.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

The evaluation engine draws on a variety of types of evidence. File format, basic syntax, execution status, expected outputs, and critical intermediate values are checked by rule checks and test cases. Domain knowledge and the task rubrics also narrow down the required surveying process. The large language model analyzes the program structure and explains the probable errors and offers tailored suggestions for revisions. Teacher review includes borderline judgments, inconsistent model outputs, unusual but valid solutions, and final confirmation. This is a design that helps reduce the likelihood of treating fluent AI-generated comments as automatically correct.

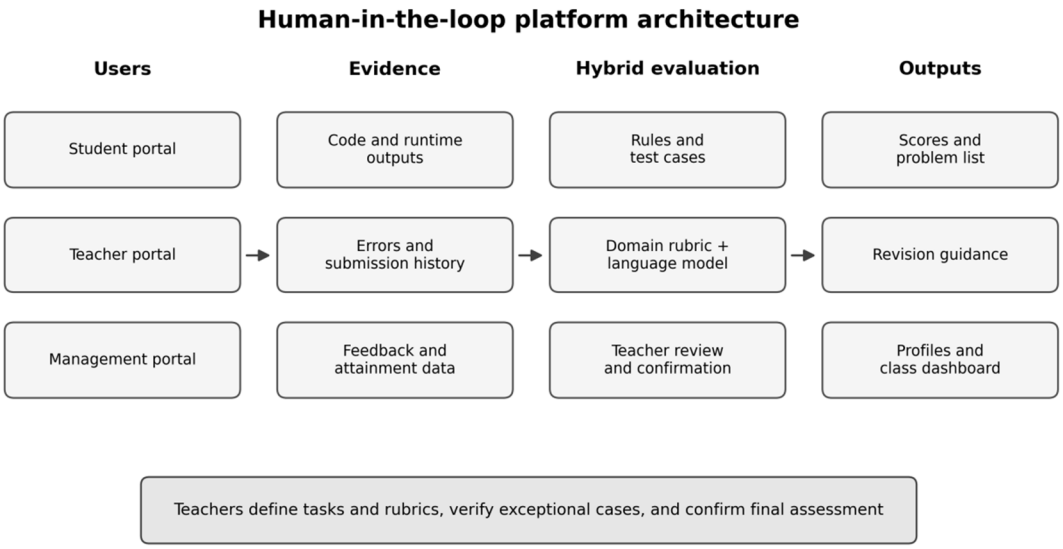

Figure 2. Human-in-the-loop platform architecture

4.3. Hierarchical Task Bank

Table 2. Hierarchical task-bank design

Task level	Representative content	Primary assessment focus
Level 1: Python foundations	Variables, conditions, loops, functions, lists, dictionaries, file I/O	Correct syntax, basic logic, readable structure
Level 2: Single surveying calculations	Coordinate increments, azimuths, distances, leveling, closure errors	Translate formulas and rules into executable steps
Level 3: Integrated data processing	Read observation files, batch computation, result-table output	Data pipeline, precision control, exception handling
Level 4: Engineering extensions	Route curves, deformation-monitoring data, GNSS-format processing	Transfer programming ability to authentic professional contexts

Each task consists of knowledge-point tags, input data, expected outputs, reference calculations, scoring dimensions, and test cases. The task bank establishes structured relationships between professional knowledge and evidence for its evaluation. For example, a closed-traverse can be subdivided into angular closure, azimuth propagation, coordinate increments, coordinate closure, correction allocation, and output to result-table. This will allow the system to identify

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

instances were, despite an apparently correct ending result, the incorrect method may be the problem.

4.4. Multi-Dimensional Code Assessment

Table 3. Six dimensions of code assessment

Dimension	Evidence examined	Guiding question
Result correctness	Final and intermediate results, required precision, key variables	Are the surveying results correct and traceable?
Algorithm logic	Calculation sequence, branching, loops, surveying business rules	Does the program implement the correct professional workflow?
Code quality	Naming, functions, comments, duplication, readability	Is the code understandable and maintainable?
Exception handling	Invalid inputs, missing files, abnormal values	Can the program respond safely to common failures?
Runtime performance	Executability, repeated reading, unnecessary loops, stability	Does the program run reliably and reasonably efficiently?
Extension and innovation	Batch processing, Excel output, visualization, reusable modules	Does the student extend the task in a meaningful way?

The dimensions are not applied in a rigid manner, but rather in a flexible manner to each assignment. The emphasis in basic tasks is shifted more toward correctness and syntax, while integrated tasks emphasize greater algorithmic completeness, exception handling, and quality of output. Additionally, the teacher can review the evidence that underlies each dimension, making the process more transparent than a single, opaque AI score.

4.5. Personalized Feedback and Resubmission

Feedback is designed to the learner's existing problem. Common errors like syntax, file path, undefined variable, data type conversion, etc. are clearly explained in the platform if the student has a weak foundation. It focuses on topics of intermediate students, such as the ordering of an algorithm, the normalization of angles, the boundaries of a loop, and the structure of a function. Extensions are recommended for stronger students: batch processing, exception handling, spreadsheet output, visualization, and reusable functions. The primary objective is to help the learner take over their thinking process by offering a full, complete solution.

Students are able to revise and resubmit their programs. This ensures that feedback can be used to make the next try more effective and allows the platform to see if the same issues occur, go away, or change across different attempts. Teachers can check the AI-generated message and then the student's revised code to distinguish between superficial and deep learning.

5. Classroom Implementation

5.1. Teaching Workflow

The classroom operations were structured based on five practical steps. The teacher recorded a surveying task and input/output requirements, key checkpoints, and an assessment rubric before class. While working on the task, students designed the algorithm and built the Python program. Once they submitted, the platform did rule checking, test case running, generated a structured code report, and provided feedback. Students then made another revision to their

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

program and resubmitted it. Last, the teacher reviewed class-level patterns and evidence of understanding to determine if she should give a class-wide explanation, a mini-lesson, or individual support.

5.2. Example: Closed-Traverse Coordinate Computation

The closed-traverse task illustrates the critical importance of being domain-aware when assessing. Well, to get it right, a solution needs to determine the angular closure, compare it with the specified tolerance, distribute the angular adjustments as needed, advance the azimuths in the correct direction, normalize angles to the specified range, compute the required coordinate increments, assess coordinate closure, make remaining adjustments, and then output the adjusted coordinates. The platform not only checks the ending coordinates, but also important intermediate results and logic in the code. Therefore, it can distinguish between a numerical formatting error, a Python loop error, and an error in the surveying method.

Feedback is provided in a "problem chain", and not as a general comment. A typical report will include the name of the code section, what went wrong, reference to a particular surveying checkpoint and what needs to be changed. If the azimuths are outside the range of 0-360, for example, then the report says that the normalization step was not performed and tells the student to check the azimuth propagation function. If the coordinate corrections are given the wrong sign, the report will point to the closure of balancing instead of merely stating that the final coordinates are incorrect.

5.3. Teacher Intervention Based on Learning Analytics

Student records are amalgamated into student and class profiles. A student profile indicates key syntax concepts, algorithm awareness, logic and planning abilities, code quality, debugging performance, task reliability, and risk level. Teachers can see mastery of knowledge points, breakdown of error types, assignment progress, score trends and ability level distribution on the class dashboard. The teachers use these indicators as diagnostic rather than automatic cues. For instance, if the same type of errors with regard to file-reading or indentation occur repeatedly, then the fundamental support can be taught again; if output is correct but the entered code is duplicated, then a function-design exercise may be appropriate; and if the same type of errors in the closure and azimuth occur repeatedly, then reteaching the process of calculating the surveying process may be warranted.

6. Initial Practice Results

6.1. Overall Dashboard Indicators

Table 4. Aggregated class-dashboard indicators

Indicator	Observed value	Interpretation boundary
Class size	42 students	Engineering Surveying Class 2401
Semester	Spring 2026	Initial platform application
Assignment completion	95%	Five recorded assignments
Overall average score	84.6 / 100	Dashboard aggregate
Students flagged for attention	4	Diagnostic prompt for teacher follow-up
Ability levels	10 excellent; 18 good; 10 medium; 4 warning	23.8%, 42.9%, 23.8%, and 9.5%

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

The dashboard shows that most students completed the tasks recorded on the dashboard and that the largest ability group was termed "good." Four students were identified for follow-up. These signs may be useful for planning teaching support but should not be considered psychological or ability traits. The categories are based on the evidence of the task that is available at this time and could be different if students do more work.

6.2. Descriptive Score Trend

The class-average scores for the five assignments were 78.1, 82.3, 84.6, 86.7, and 88.9. The difference between the first and 5th assignment was 10.8 points. This is the upward trend of receiving feedback, revising work, and getting to know the tasks more. The available records, however, do not distinguish between the effect of AI feedback and other variables, like typical practice, teacher instruction, task difficulty, student adaptation, and repeated exposure. Therefore, this figure is intended as an indicator of implementation and not as a causal estimate.

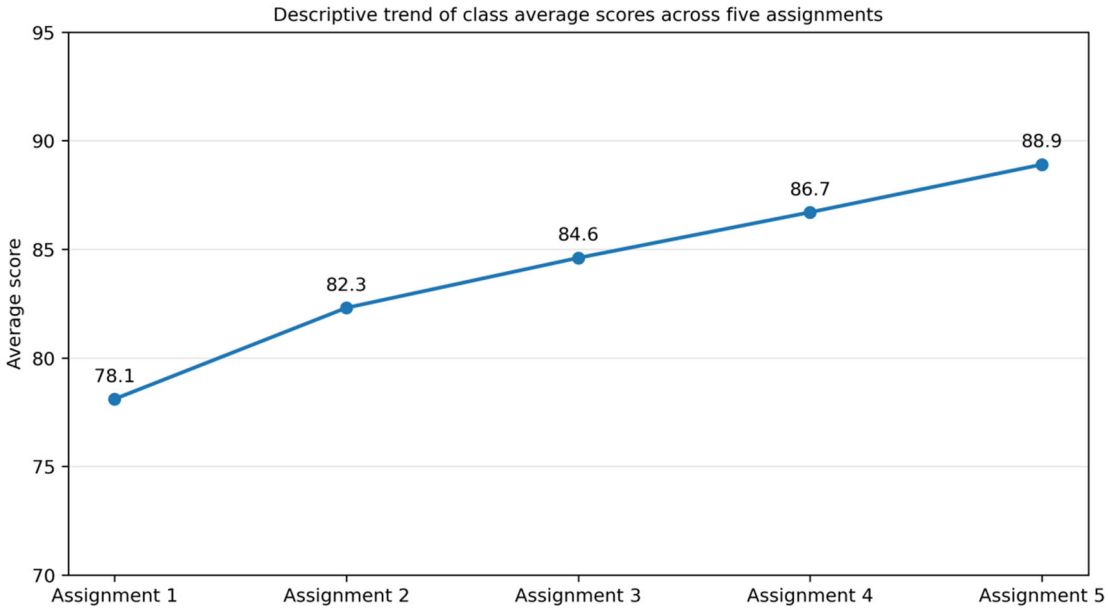

Figure 3. Descriptive trend of class-average scores across five assignments

6.3. Knowledge Mastery and Error Patterns

Table 5. Class-level mastery indicators

Knowledge/ability indicator	Dashboard mastery rate
Python foundations	92%
File input/output	88%
Loops and conditions	85%
Surveying algorithm logic	82%
Exception handling	76%
Standardized result output	90%

They had the lowest mastery rate of exception handling (76%) and the next was the logic for surveying (82%). In general, students tended to be more at ease with basic syntax and standardized output than with more robust engineering practices and the entire, professional reasoning chain. The Error Distribution also provides a more evident foundation for diagnosis in terms of what to teach. The exceptions encountered while trying to read the file represented

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

28% of all the errors recorded, and the same applied to the non-standard variable names, which accounted for 22%. The loop-boundary errors accounted for 18%, the result-type conversion errors 14%, the missing exception handling errors 10%, and the inconsistent output format errors 8%.

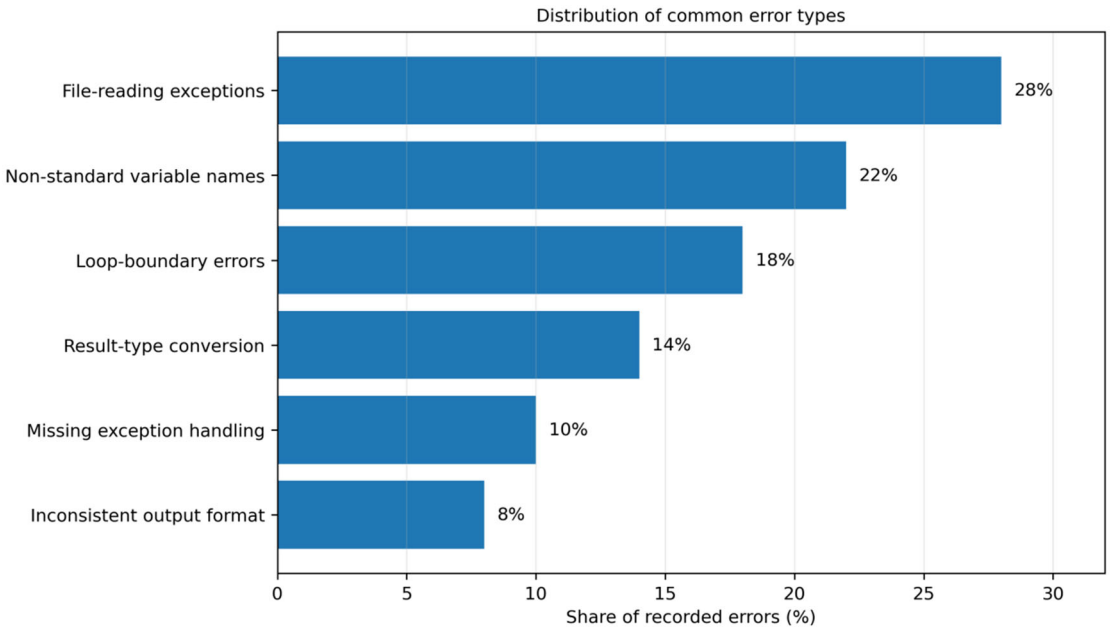

Figure 4. Distribution of common recorded error types

An average score is not as useful for instruction as mastery indicators plus specific error categories. It stresses 3 aspects of the next iteration of teaching: a focus on file-based data pipelines, explicit teaching of code readability and naming as professional skills, and the connection between the design of loops and the structure of surveying observations. It also suggests that exception handling should be built into typical surveying work, instead of being delayed until a later, more advanced programming topic.

6.4. Observed Instructional Value

The platform transformed the first phase of the grading process into a teacher. Instead of going through each program and beginning at the start, the teacher was given a report that provided feedback, evidence for each dimension, error locations, and a draft report. This allowed the teacher to spend more time on the most important calculations, on special cases, and on more individualized teaching. It also ensured the consistency of dimensions in the assessment and a clear record, while supporting the traceability of tasks.

Feedback was more than a numeric grade from the student's point of view. Immediate feedback on errors and specific feedback on revision provided a starting point for debugging, and extension suggestions for stronger students. The resubmission feature fostered an iterative programming process: learning the task, designing an algorithm, writing the code, testing, revising and documenting results. The aggregated evidence also highlighted class-level issues early in the lesson and informed the teaching of lessons later in the class.

7. Discussion

7.1. From Automated Grading to Formative Learning Support

It's not just about replacing teachers with AI,the model's primary benefit lies in this. Rather, it is about the connection between assessment evidence and learner action and the support of

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

teacher decision making. Deterministic tests offer reliable evidence for outcomes that can be substantiated, generative AI makes difficult problems more comprehensible for guidance learners, re-submission allows the guidance learner to follow up on feedback received, and learning analytics allows the teacher to have an aggregated evidence base. The overall design is consistent with formative design theory, as the feedback is intended to help close the performance gap and not just report it [4-7].

7.2. From a General-Purpose Chatbot to a Professional Teaching Tool

A general chatbot may be able to explain Python, but it cannot guarantee to understand the specific surveying procedure to be followed, the expected intermediate values, the local constraints on the task, or the assessment criteria to be applied. The design here does this by making knowledge of the domain explicit by task decomposition, tags, reference calculations, test cases, and rubrics. This is very important in geomatics education as a program may be syntactically correct but still be a professional error. For this reason, the hybrid architecture is suitable for other vocational courses involving data-intensive applications, such as processing GNSS data, engineering measurement, deformation monitoring, processing of remote-sensing images, spatial analysis in GIS, and processing data from unmanned-aerial-vehicles.

7.3. Data-Informed Course Improvement

Normally random classroom artifacts become a longitudinal evidence base on the platform. No more separate Code, Runtime Results, Error Logs, Revisions and Feedback – all of these can now indicate which Knowledge Points are challenging, which tasks result in non-productive struggle, and which assessment criteria should be clarified. This evidence can help the course team to redesign tasks, to plan lessons together, to review the progress of the course's targets, to select competitions, and to deliver targeted project training. This in turn extends the benefits of formative assessment beyond providing individual feedback to program-level quality improvement.

7.4. Risks, Governance, and Limitations

Four key risks call for ongoing governance. Firstly, the output of the AI might not be correct or consistent, especially if the code is not complete or more than one valid algorithm is possible. As a result, teacher verification and deterministic evidence are a must. Second, feedback may reveal too much of the answer and diminish the benefit of struggling. Prompts and templates should be used to encourage questions, hints and revision guidance but not to give any replacement code. Thirdly, students can become too reliant on the AI or provide code that they cannot explain. To compensate for this, the following methods of oral questioning, process documentation, in-class coding, and explanation requirements should be combined with platform-based assessment. Fourth, source code and learning profiles are data in education. From this perspective, access control, limited access to the minimum necessary, anonymized reporting of the research, and secure deployment are required, in accordance with responsible-AI principles [1, 3, 11-13].

There are also obvious methodologic shortcomings in this study. It is targeted at one class, one semester, five assignments, and consolidated dashboard metrics. No control group, no pretest–posttest design, no validated student questionnaire, no independent review of the quality of feedback, and no follow-up. It is possible that the difficulty of the task varied, and that the score growth observed may be due to multiple factors. Further studies should include comparisons between cohorts or conditions, evaluate the degree of agreement between AI and teacher evaluations, examine uptake of feedback, examine code quality changes, collect student perceptions, and examine transfer of improvements to new engineering tasks.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

8. Conclusion

This study presents a Generative AI based formative assessment approach for a higher vocational course in Surveying Programming (Python). It integrates task-specific task design, multiple sources of process data, deterministic checks, domain-specific rubrics, language model interpretation, personalized feedback, opportunities for resubmission, learning analytics, and teacher verification. The system generated usable evidence at the individual student, class, and course levels when it was first adopted in a class of 42 students. The descriptive dashboard showed high rates of task completion, improving average trends for assignments, and clear trends for file handling, file naming, loop design, surveying reasoning, and exception handling.

The bottom line is that generative AI is truly educational when it is integrated into a formal assessment process and not simply an open-ended answer generator. The model here suggested serves as a practical foundation for the application of data for timely feedback and for teaching decisions in surveying programming. More extensive multi-class and multi-year studies are needed to validate its effectiveness, fairness, and transferability to other contexts.

References

[1] Ministry of Education of the People's Republic of China, National Development and Reform Commission, Ministry of Industry and Information Technology, Ministry of Science and Technology, & National Data Administration. (2026). Artificial intelligence + education action plan (2026). [In Chinese]

[2] CPC Central Committee & State Council. (2025). Education powerhouse development plan (2024-2035). [In Chinese]

[3] UNESCO. (2023). Guidance for generative AI in education and research. UNESCO.

[4] Black, P., & Wiliam, D. (1998). Assessment and classroom learning. Assessment in Education: Principles, Policy & Practice, 5(1), 7-74. https://doi.org/10.1080/0969595980050102

[5] Hattie, J., & Timperley, H. (2007). The power of feedback. Review of Educational Research, 77(1), 81-112. https://doi.org/10.3102/003465430298487

[6] Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. Studies in Higher Education, 31(2), 199-218. https://doi.org/10.1080/03075070600572090

[7] Shute, V. J. (2008). Focus on formative feedback. Review of Educational Research, 78(1), 153-189. https://doi.org/10.3102/0034654307313795

[8] Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education, 19(1), Article 3. https://doi.org/10.1145/3231711

[9] Messer, M., Brown, N. C. C., Kölling, M., & Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. ACM Transactions on Computing Education, 24(1). https://doi.org/10.1145/3636515

[10] Nguyen, H., & Allan, V. (2024). Using GPT-4 to provide tiered, formative code feedback. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education* (Vol. 1, pp. 958-964). https://doi.org/10.1145/3626252.3630960

[11] Kasneci, E., Sessler, K., Küchemann, S., et al. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 103, 102274. https://doi.org/10.1016/j.lindif.2023.102274

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202608_9(8).0003

[12] Tlili, A., Shehata, B., Adarkwah, M. A., et al. (2023). What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. Smart Learning Environments, 10, Article 15. https://doi.org/10.1186/s40561-023-00237-x

[13] Lo, C. K. (2023). What is the impact of ChatGPT on education? A rapid review of the literature. Education Sciences, 13(4), 410. https://doi.org/10.3390/educsci13040410

[14] Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education-where are the educators? International Journal of Educational Technology in Higher Education, 16, Article 39. https://doi.org/10.1186/s41239-019-0171-0

[15] Li, X. Z., Zhang, Y. X., Xie, J. J., Zhang, R., Liu, M. M., & Zhao, J. (2026). Teaching practice of explainable feedback in programming empowered by large language models. International Journal of Social Science and Education Research, 9(7), 60-67.

[16] Yang, J., Zhang, Y. X., Xie, J. J., Chen, Y. Q., & Fu, C. T. (2026). Research on an AI-empowered three-stage closed-loop teaching model for chemical engineering practical training. International Journal of Social Science and Education Research, 9(7), 68-75.

[17] Li, Y. X. (2026). Research and practice on the teaching reform of programming courses enabled by AI: A case study of "Python fundamentals and applications". International Journal of Social Science and Education Research, 9(7), 108-115.