



AI-SOLO: A hybrid framework for cognitive assessment in programming education

A. Sangeetha^{1*}, B. Suthan¹, P. Vishaliny¹ and K. Gowrisha¹

¹*Department of Information Technology, Faculty of Computing SLIIT Northern university*

Abstract

Traditional programming assessments focus on correctness and efficiency and do not consider students' cognitive learning processes or code structure. Teachers then encounter challenges in providing effective feedback to facilitate cognitive learning in programming education. This study proposes a hybrid AI framework, AI-SOLO, that combines CodeBERT semantic embeddings, Graph Neural Networks (GNNs), and behavioral metadata to automatically categorize student programming submissions. Each submission was mapped to the relevant level of understanding described in the Structure of Observed Learning Outcomes (SOLO) Taxonomy. The framework is multi-dimensional, combining semantic, structural and behavioral evidence of understanding, rather than just output correctness. It was tested in a controlled study with 120 undergraduate students who submitted 1,824 Python programs. AI-SOLO achieved an overall classification accuracy of 91.4% and 83.4% at the Extended Abstract level, demonstrating its ability to classify programming submissions across cognitive levels. The framework also enables immediate, personalized feedback to learners, targeted instructional interventions, and interpretable dashboards for educators to monitor cognitive progress at the class level. The findings indicate that integrating semantic, structural, and behavioral features can support the cognitive-level classification of student programming submissions and provide a basis for structured assessment in programming education.

Keywords: SOLO Taxonomy, Artificial Intelligence, Cognitive Assessment, Graph Neural Networks, CodeBERT

Article info

Article history:

Received 20th June 2026

Received in revised form 15th July 2026

Accepted 25th August 2026

Available online 20th September 2026

ISSN (E-Copy): ISSN 3051-5262

ISSN (Hard copy): ISSN 3051-5602

Doi: <https://doi.org/10.4038/jmtr.v11i4.360>

ORCID iD: <https://orcid.org/0009-0002-1802-2585>

*Corresponding author:

E-mail address: sangeetha.b@slit.lk (A. Sangeetha)

[CC BY-NC-SA](#) © 2026, The Author(s)

Introduction

Programming assessment is a topic of increasing significance as computer science programs expand in size and scope, and scalable, accurate, and pedagogically meaningful approaches to programming assessment are needed. Traditional assessment techniques such as unit testing, execution-based evaluation, and code quality metrics focus on determining if a program generates the desired results and runs efficiently. These methods, however, offer only a superficial understanding of students' knowledge, such as the way they structure programming concepts, build logical connections, and use computational reasoning (Harimurti et al., 2023). This can be difficult for teachers to identify different levels of thinking and to provide feedback appropriate to students' understanding and use of programming knowledge.

Cognitive taxonomies provide theoretically grounded approaches for characterizing different levels of learning. Bloom's Taxonomy is one of the most commonly used frameworks in educational assessment and categorizes cognitive processes into six levels: Remember, Understand, Apply, Analyze, Evaluate, and Create (Bloom, 1956). Bloom's Taxonomy has been applied in computer science education to create learning outcomes, to develop programming tasks, and to categorize the level of cognitive demand (Berglund & Lister, 2010; Thompson et al., 2008). The direct application of Bloom's Taxonomy to automated assessment of programming submissions can be difficult, however, because a single programming artefact can represent multiple overlapping cognitive processes. For instance, when the assessment is based mainly on inferred cognitive processes, as in writing or modifying a program, it is difficult to classify the program consistently and automatically (Whalley et al., 2011).

Another model is the Structure of Observed Learning Outcomes (SOLO) Taxonomy by Biggs and Collis (1982). SOLO describes the structural complexity of demonstrated understanding in five levels: Pre-structural, Unistructural, Multi-structural, Relational, and Extended Abstract. SOLO is not just about what the learner is supposed to think, but about the organization and complexity of the learner's response. This response-oriented view is especially important for programming assessment, as source code offers observable evidence of conceptual understanding, logical organization, relationships between programming constructs, and abstraction. Progressions across SOLO levels can then be understood as increasingly integrated and generalized structures in students' programming solutions.

With the development of artificial intelligence, automated programming assessment has evolved beyond the simple correctness-based evaluation. Machine-learning methods have been used to model students' knowledge and programming performance (Piech et al., 2015), and transformer-based models have been used to provide more semantically rich representations of source code. Feng et al. (2020) proposed CodeBERT, a pre-trained model that learns representations from both programming and natural languages for semantic analysis of code. To represent data-flow information, GraphCodeBERT extended code representation by incorporating data-flow information, emphasizing the need to explicitly represent structural relationships in programs. Graph Neural Networks (GNNs) can also be used to learn relationships in graph-based program representations, such as Abstract Syntax Trees (ASTs) and Program Dependency Graphs (PDGs) (Guo et al., 2021; Pan et al., 2025). These developments provide the potential to integrate semantic and structural information to provide a more complete analysis of student programming submissions.

Cognitive taxonomies have also been developed for use in programming education. Previous research has explored how SOLO can be used to describe novice programmers' understanding and how the structure of their responses relates to their programming performance (Lister et al., 2006a; Lister et al., 2006b). Subsequent research has continued to apply SOLO in programming education. For instance, Malik et al. (2021) examined the alignment of teaching and assessment with SOLO in a computer programming course, while Bennett and Izu (2025) applied a SOLO approach to students' ability to improve code efficiency. These studies demonstrate the value of SOLO for analyzing programming-related learning and highlight the potential for automated analysis of programming artefacts.

Meanwhile, AI-based programming assessment increasingly incorporates semantic, structural, and process-oriented information. Previous studies have examined automated programming assessment, student-performance modelling, AI-assisted tutoring, and cognitive classification (Arunprakash et al., 2026; Harimurti et al., 2023; Lytvynova et al., 2024; Piech et al., 2015). ; However, these lines of inquiry have largely remained separate. Performance-based systems tend to focus on correctness or prediction, whereas taxonomy-based systems often rely on manually defined assessment criteria or educational interpretation. Therefore, there remains a need to investigate whether semantic properties of source code, program structure, and behavioral evidence of problem-solving can be integrated within a single AI system aligned with SOLO.

Selected previous studies related to cognitive taxonomies, AI-supported programming assessment, and code-based learning analysis are summarized in Table 1. These studies demonstrate advances in automated assessment and cognitive classification, as well as the need for a framework that integrates various forms of programming evidence. The level of understanding demonstrated in programming submissions could be evaluated using a multidimensional approach, not just by checking the correctness of the output or a single representation of the source code.

The present study proposes a hybrid artificial intelligence framework, called AI-SOLO, for assessing the cognitive complexity of student programming submissions using the SOLO Taxonomy, as summarized in Table 1. The framework combines three complementary sources of evidence: semantic representations generated with CodeBERT, structural representations extracted from ASTs and PDGs and analyzed with GNNs, and behavioral metadata capturing students' problem-solving processes. By integrating these dimensions, AI-SOLO shifts the focus from correctness-only evaluation toward observable levels of structural and conceptual understanding in programming.

This study makes four main contributions. First, it operationalizes the SOLO Taxonomy through observable characteristics of student programming submissions. Second, it develops a hybrid AI architecture that integrates semantic, structural, and behavioral evidence for SOLO-level classification. Third, it evaluates the proposed framework using held-out classification metrics to examine its effectiveness across the five SOLO levels. Finally, it explores the potential of SOLO-aligned classification to support structured formative feedback and educator interpretation of students' cognitive development. Together, these contributions examine how cognitive learning theory can be integrated with multidimensional AI-based code analysis for programming assessment.

Table 1: Summary of Related Works and Research Gap

Author(s) and Year	Approach	Taxonomy Used	Limitation
Piech et al.(2015)	ML-based student performance prediction	None	No cognitive mapping; correctness only
Thompson et al.(2008)	Manual Bloom's rubric for CS curricula	Bloom's	Not scalable; no automation
Wang et al.(2021)	Process-based AI assessment framework	None	No taxonomy alignment; lacks SOLO
Lytvynova et al.(2024)	LLM-assisted programming tutoring	Bloom's (partial)	No SOLO; limited cognitive depth assessment
Harimurti et al.(2023)	Automated programming problem-solving assessment	None	No taxonomy integration

Methodology

In this study, a design-based experimental approach was adopted to develop and evaluate a hybrid AI framework for cognitive assessment in programming education. The framework is based on the SOLO Taxonomy, which classifies student programming submissions by the depth and structure of understanding demonstrated in the code. The proposed methodology is different from the traditional assessment, as it takes into account three types of evidence: semantic, structural and behavioral, and integrates them to provide a more comprehensive assessment of students' cognitive performance.

During one academic year, 120 undergraduate students from introductory and intermediate Python courses at SLIIT Northern University (Jaffna Branch) participated in the study and submitted programming solutions via Replit and HackerRank for Education. Students attempted 10 to 25 problems from a problem bank designed to cover all five SOLO levels. The source code and behavioral metadata generated by the platform were retained for each submission to support analysis. After preprocessing, 1,824 unique programming submissions from 120 students remained for analysis. The distribution of the labelled programming submissions across the five SOLO Taxonomy levels is presented in Table 2.

Table 2: Distribution of Programming Submissions across SOLO Levels

SOLO Level	Number of Submissions	Percentage
Pre-structural	186	10.2%
Unistructural	368	20.2%
Multi-structural	542	29.7%
Relational	476	26.1%
Extended Abstract	252	13.8%

We employed CodeBERT, a pre-trained transformer model for programming code and natural language, to generate contextual semantic embeddings for each program. These embeddings support conceptual analysis and help distinguish structurally similar but semantically different submissions. CodeBERT was selected as the semantic encoder so that structural dependencies could be modelled separately through AST- and PDG-based GNN components. This separation reduces unnecessary overlap between semantic and structural representations and allows the two modalities to provide complementary evidence within the hybrid framework. Programs were translated into Abstract Syntax Trees (ASTs) for structural analysis and Program Dependency Graphs (PDGs) for dependency analysis. These representations were then fed into Graph Neural Networks (GNNs) to learn higher-order patterns, such as connectivity, modularity, and structural complexity, which correspond to SOLO levels. Indicators for behavioral analysis were derived from platform logs, including revision cycles, error-correction patterns, and time on task.

These features reflect students' problem-solving processes and differentiate similar-looking submissions that vary in the level of cognitive engagement required. The semantic embedding generated by CodeBERT, the structural embedding produced by the GNN, and the normalized behavioral feature vector were first projected into compatible feature dimensions using fully connected layers. The three feature representations were then concatenated to form a unified multimodal feature vector. This fused representation was passed through a multilayer perceptron (MLP) classification head consisting of fully connected layers, ReLU activation, dropout regularization, and a final Softmax output layer that generated probabilities for the five SOLO classes. Model performance was assessed in terms of accuracy, precision, recall, F1 score, and confusion matrix analysis, evaluated separately for each SOLO level for fine-grained assessment.

The dataset was partitioned at the student level into training (70%), validation (15%), and test (15%) sets. All submissions from the same student were retained in a single partition; therefore, no student's submissions appeared in more than one dataset split. This student-level grouped partitioning was used to prevent information leakage arising from multiple submissions produced by the same learner. The model was trained using the Adam optimizer (learning rate 1×10^{-4} , weight decay 1×10^{-5}) for 20 epochs, with a batch size of 32. Cross-entropy loss was weighted by inverse class frequency to mitigate class imbalance. Early stopping, with patience 5, was applied to the validation macro-F1 score to prevent overfitting. All experiments used a fixed random seed (42) for reproducibility and were run on an NVIDIA RTX 3090 GPU with PyTorch 2.1.0.

A labelled dataset was developed by assigning submissions to the five SOLO levels. The annotation process was based on a programming-specific SOLO rubric covering conceptual correctness, use of programming constructs, logical organization, and abstraction. The operational mapping of the five SOLO levels to observable programming characteristics is presented in Table 3. The submissions were independently annotated by three computer science educators with experience in programming education, assessment, and curriculum development. Before annotation, the raters were familiarized with the programming-specific SOLO rubric, including definitions, representative coding examples, and calibration exercises covering all five SOLO levels. Each submission was independently evaluated according to the predefined SOLO criteria. Cases of disagreement were reviewed through structured discussion among the raters until consensus was reached.

Table 3: Programming-specific SOLO Taxonomy Mapping Rubric

SOLO Level	Observable Programming Characteristics
Pre-structural	Submission is missing, irrelevant, syntactically invalid, or demonstrates no meaningful understanding of the programming task.
Unistructural	Uses one relevant programming concept or construct correctly but does not integrate it with other required elements.
Multi-structural	Uses several relevant constructs correctly, such as variables, conditional statements, loops, or functions, but these remain largely independent or weakly connected.
Relational	Integrates multiple programming constructs into a coherent solution in which control flow, data handling, functions, and logical components work together to solve the problem.
Extended Abstract	Demonstrates abstraction, generalization, reusable functions or classes, optimization, alternative solution strategies, or adaptation of the solution beyond the immediate task requirements.

The trained hybrid model was deployed as part of a web-based assessment platform that exposes the predicted SOLO level and confidence score, generates formative feedback for students aligned to SOLO levels, and provides an educator dashboard that presents the class distribution across SOLO levels, cognitive growth indicators, and evidence drawn from semantic importance, graph metrics, and behavioral patterns. Proposed AI-SOLO Framework is illustrated in Figure 1.

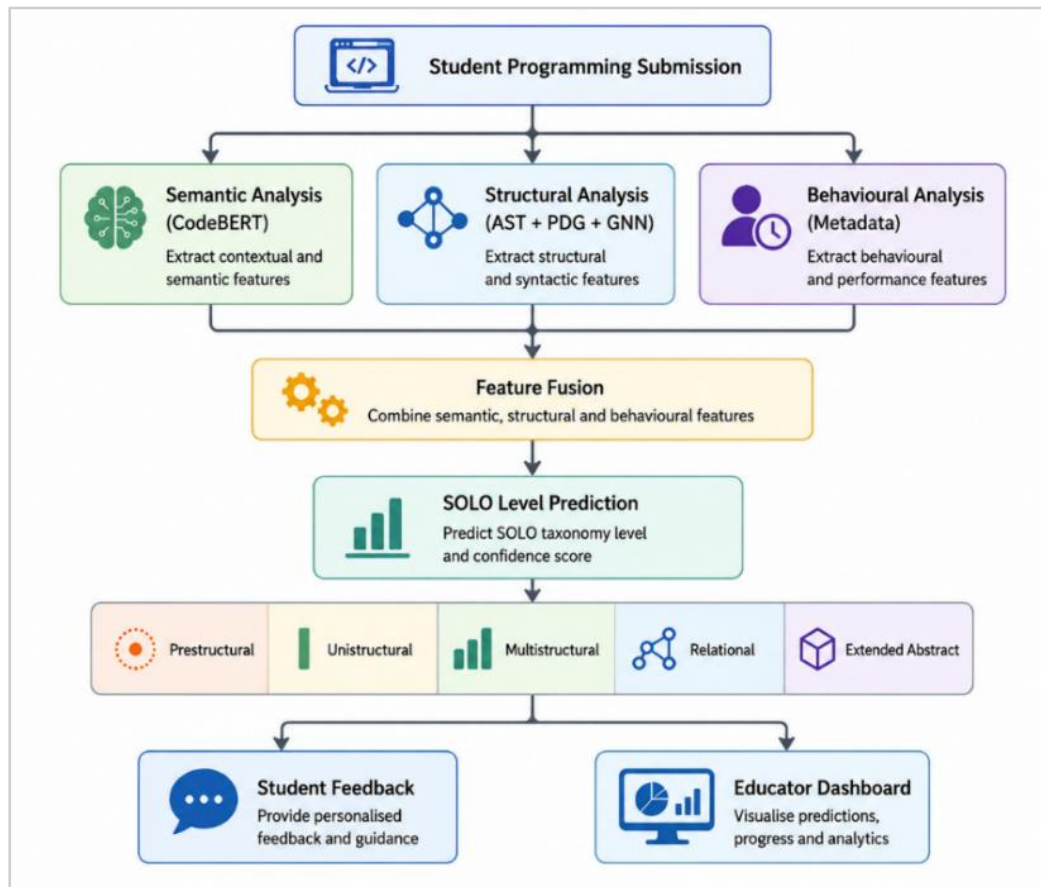


Figure 1. AI-SOLO Framework

Ethical Clearance Statement

Ethical approval was not required for this study, as it is based on the analysis of student coursework submissions collected during regular instructional activity. Nevertheless, established ethical research practices were strictly followed. All student participants provided informed consent prior to data collection, and participation was entirely voluntary and unrelated to course grades. A written information sheet was given to participants explaining the study, the types of data that would be collected, how the data would be used, and that they could withdraw at any time without penalty. All data were anonymized before analysis: student names, identifiers and institutional affiliations were replaced with non-reversible identifiers. The study was carried out in compliance with the Declaration of Helsinki (as amended) and the data protection policies of the host institution.

Evaluation of the framework focused on quantitative classification performance using accuracy, precision, recall, F1-score, and confusion-matrix analysis across the five SOLO levels. These measures were used to assess the technical performance of AI-SOLO on the held-out evaluation data. The evaluation design maintained student-level separation across training, validation, and test partitions to support a controlled assessment of classification performance.

Results

As shown in Figure 2, AI-SOLO achieved high classification accuracy at all five SOLO levels. The model's performance was highest at the Pre-structural level, with an accuracy of 97.2%, indicating high reliability in detecting incoherent or irrelevant answers. The accuracy of the Unistructural level was 94.5%, while the Multi-structural and Relational levels were 92.8% and 89.1%, respectively. A moderate decrease in accuracy was observed at higher cognitive levels and with more complex and abstract tasks, with the Extended Abstract level at 83.4%, the most complex and abstract level.

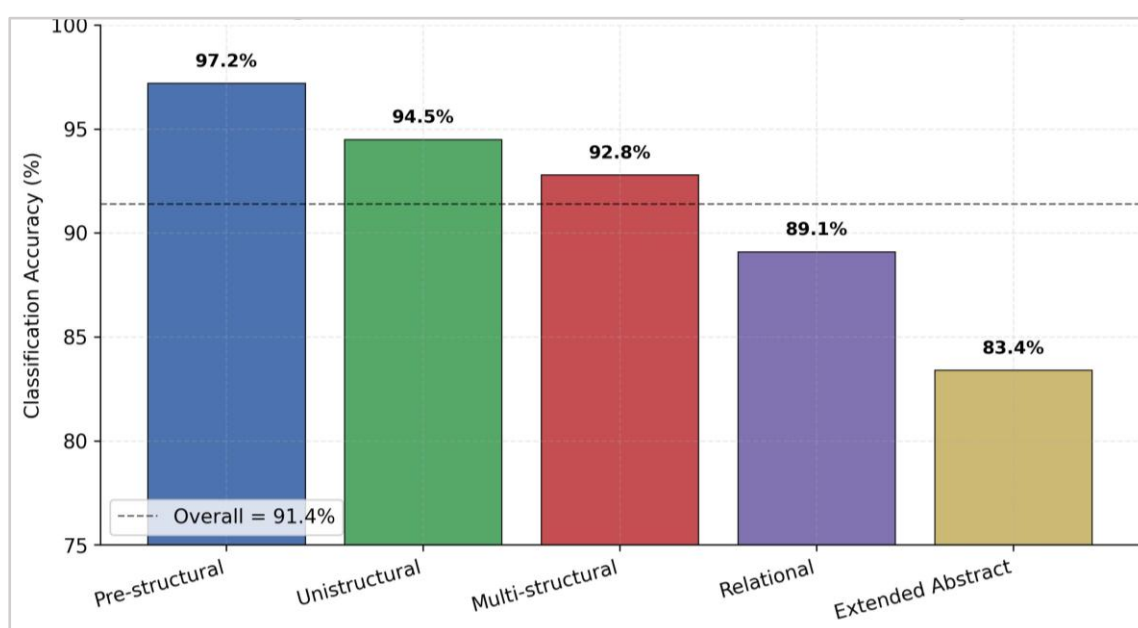


Figure 2. Classification Accuracy across SOLO Levels

The test-set confusion matrix is shown in raw counts (left) and row-normalized (right) in Figure 3. There are two pedagogically significant observations. First, most misclassifications are between adjacent SOLO levels, not between distant ones, such as the largest off-diagonal mass in the Extended Abstract row is the confusion with the immediately adjacent Relational level. This monotonic confusion pattern aligns with the structural ordering of the SOLO Taxonomy and the proposed mapping. Second, very few Pre-structural submissions are misclassified as Relational or Extended Abstract, the failure mode that would have the most negative impact on formative feedback.

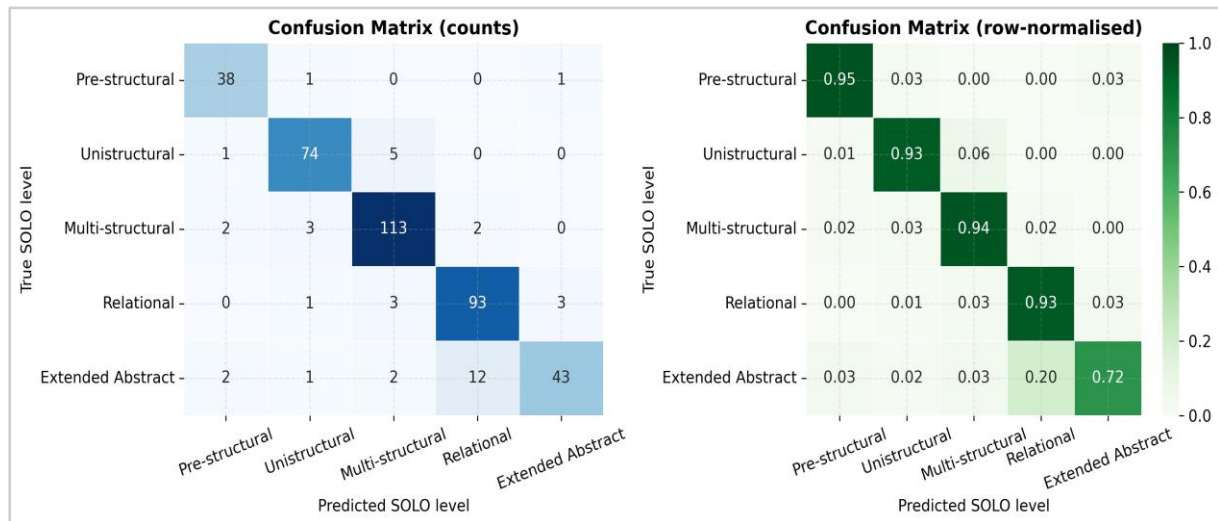


Figure 3. Confusion Matrix on the Held-out Test Set

In addition to overall accuracy, precision, recall, and F1-score were calculated for each SOLO level to provide a more comprehensive evaluation of classification performance. For the Pre-structural level, the model achieved a precision of 88.4%, a recall of 95.0%, and an F1-score of 91.6%. The Unistructural level achieved 92.5% precision, 92.5% recall, and a 92.5% F1-score, while the Multi-structural level achieved 91.9% precision, 94.2% recall, and a 93.0% F1-score. For the Relational level, precision, recall, and F1-score were 86.9%, 93.0%, and 89.9%, respectively. The Extended Abstract level achieved a precision of 91.5%, a recall of 71.7%, and an F1-score of 80.4%, indicating that although predictions at this level were generally precise, a proportion of Extended Abstract submissions were misclassified, mainly as Relational. Across all five SOLO levels, the macro-averaged precision, recall, and F1-score were 90.2%, 89.3%, and 89.5%, respectively, while the weighted-average precision, recall, and F1-score were 90.4%, 90.3%, and 90.1%.

The training and validation dynamics over 20 epochs are shown in Figure 4. The training and validation accuracy curves converge smoothly, and the final validation accuracy reaches 91.4% at epoch 20. The relatively small gap between the two curves suggests no clear evidence of substantial overfitting. The convergence pattern indicates stable learning across epochs, while the class-level results reported above show that the Extended Abstract level remained the most challenging category.

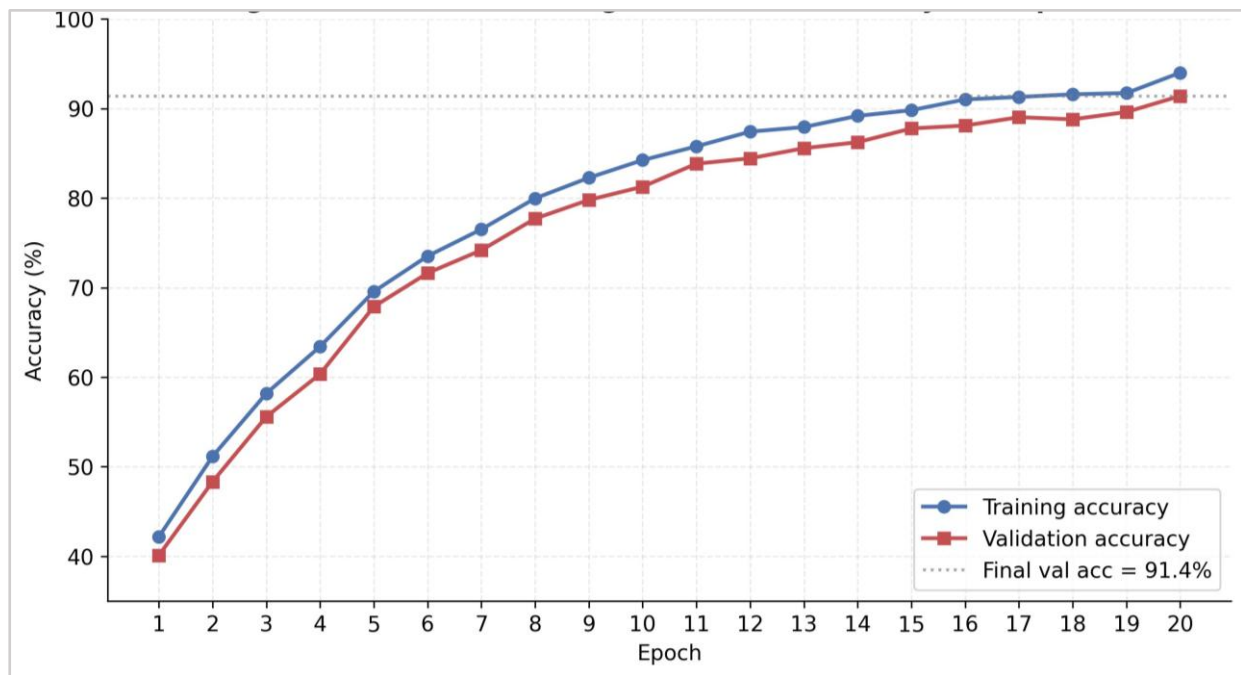


Figure 4. Training and Validation Accuracy across Epochs

Discussion

The results indicate that the hybrid AI-SOLO model captures distinct patterns across the five SOLO levels. Performance was stronger at lower levels and decreased at the Extended Abstract level, suggesting that classification becomes more challenging as programming responses require greater integration, abstraction, and generalization. The combination of semantic, structural, and behavioral features provides complementary evidence of student understanding rather than relying solely on output correctness. This pattern supports the use of SOLO as a response-oriented taxonomy for analyzing observable structure in programming artefacts.

The focus on observable response structure is a natural fit with the artefacts of programming in SOLO. The framework learned hierarchical and modular patterns in code by combining AST and PDG representations with GNNs, and semantic embeddings from CodeBERT captured conceptual nuance. Behavioral metadata added further discriminative power by providing insight into the processes students used to solve problems, allowing the framework to differentiate students who reached similar outputs via different cognitive pathways.

The framework provides actionable learning opportunities. SOLO-aligned formative feedback can help teachers plan interventions based on each learner's current level of understanding, while the educator dashboard supports class-level monitoring and the identification of students who may require additional support. This combination addresses a longstanding challenge in conventional assessment: providing individualized feedback while maintaining a broader view of class-level cognitive progress.

Although the framework demonstrated strong classification performance, several methodological limitations should be acknowledged. First, the study involved 120 students from

a single institutional context, which may limit the generalizability of the findings to other institutions, educational levels, programming languages, and curricula. Although 1,824 programming submissions were analyzed, these submissions were generated by the same 120 students and therefore were not treated as independent observations; student-level partitioning was used to minimize data leakage across the training, validation, and test sets. Second, the distribution of submissions across the five SOLO levels was imbalanced, particularly at the Pre-structural and Extended Abstract levels. Although inverse-frequency class weighting was applied, the smaller number of examples at these levels may have affected classification reliability. Third, SOLO-level annotation depends partly on expert interpretation, and despite the use of a predefined programming-specific rubric and independent educator assessment, some subjectivity may remain in distinguishing adjacent levels such as Multi-structural and Relational. Fourth, the behavioral features were limited to information available from the programming platforms, including revision cycles, error-correction patterns, and time on task, and therefore did not capture off-platform activities, collaboration, or external assistance. Finally, the framework was evaluated primarily on introductory and intermediate Python programming tasks and was not externally validated using an independent dataset from another institution. Future studies should therefore evaluate AI-SOLO across larger and more diverse student populations, programming languages, institutional settings, and independent datasets to establish its broader validity and generalizability.

Conclusions

This study proposed AI-SOLO, a hybrid framework for classifying student programming submissions according to the SOLO Taxonomy by integrating semantic embeddings (CodeBERT), structural analysis (AST + PDG + GNN), and behavioral metadata. The framework demonstrated strong classification performance across the five SOLO levels and showed the potential of multidimensional evidence to capture differences in conceptual understanding and structural complexity that are not represented by output correctness alone. SOLO-aligned assessment can support personalized formative feedback and help educators track cognitive progress at both individual and class levels. The framework also links automated cognitive assessment with interpretable educator dashboards, connecting technical analysis with pedagogical decision-making. Because the study involved 120 students from a single institutional context and focused on introductory and intermediate Python programming tasks, broader claims should be made with caution until AI-SOLO is validated across larger, more diverse, multi-institutional, and cross-language datasets.

Acknowledgements

The authors thank the participating students for their programming submissions and the panel of computer science educators who carried out the SOLO annotation. We also thank the colleagues who provided constructive comments on earlier drafts of this manuscript.

Declaration of Funding Sources

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of interest statement

The author declares no conflict of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request. Raw student submissions cannot be shared due to participant consent and institutional data-protection constraints; a synthetic, representative subset together with the code and trained-model checkpoint will be released under a permissive open-source license to support reproducibility.

Author Contribution

All authors accepted responsibility for the content of this paper, reviewed the findings, and approved the final version for submission.

Conceptualization and study design: A.S., B.S.; theoretical framework development (SOLO Taxonomy mapping and AI integration): A.S., B.S.; methodology design (hybrid CodeBERT, GNN, and behavioral metadata pipeline): B.S., P.V.; data collection, platform setup, and SOLO-based annotation: P.V., K.G.; feature extraction and model development: B.S., P.V.; model training, validation, and evaluation: B.S.; interpretation of results and pedagogical implications: A.S., B.S.; writing original draft preparation: B.S., P.V.; writing — reviewing and editing: A.S., K.G.; supervision: A.S.; project administration and coordination: A.S.

References

- Arunprakash, N., Arunpirakash, S., & Pirathap, V. (2026). *AI-driven cognitive assessment in programming education: A Bloom's Taxonomy-aligned framework for skill development* [Conference paper]. 17th International Conference on Sustainable Built Environment 2026 (ICSBE 2026).
- Bennett, A., & Izu, C. (2025). Replicating a SOLO approach to measure students' ability to improve code efficiency. In *Proceedings of the 2025 ACM Global Computing Education Conference (CompEd 2025)*. Association for Computing Machinery.
<https://doi.org/10.1145/3736181.3747166>
- Berglund, A., & Lister, R. (2010). Introductory programming and the didactic triangle. In *Proceedings of the 12th Australasian Computing Education Conference (ACE 2010)* (pp. 35–44). Australian Computer Society.
- Biggs, J. B., & Collis, K. F. (1982). *Evaluating the quality of learning: The SOLO taxonomy (structure of the observed learning outcome)*. Academic Press.
- Bloom, B. S. (Ed.). (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. Longmans, Green.
- Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 1536–1547).

- Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.findings-emnlp.139>
- Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., & Zhou, M. (2021). GraphCodeBERT: Pre-training code representations with data flow. *arXiv*. <https://doi.org/10.48550/arXiv.2009.08366>
- Harimurti, R., Anistyasari, Y., Rusimamto, P. W., & Haryudo, S. I. (2023). Automatic programming assessment to measure programming problem-solving skills. In *Proceedings of the 4th Annual Conference of Engineering and Implementation on Vocational Education (ACEIVE 2022)*. EAI. <http://dx.doi.org/10.4108/eai.20-10-2022.2328840>
- Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006a). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2006)* (pp. 118–122). Association for Computing Machinery. <https://doi.org/10.1145/1140124.1140157>
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., Marshall, L., McCauley, R., McKenna, J., Prasad, C., Sanders, K., & Simon, B. (2006b). Roundtable panel: Concepts in novice programmers. In *Proceedings of the 11th annual conference on Innovation and Technology in Computer Science Education (ITiCSE 2006)* (p. 330). Association for Computing Machinery.
- Lytvynova, S. H., Rashevskaya, N., & Proskura, S. (2024). The use of artificial intelligence in teaching students programming languages. In S. Papadakis (Ed.), *Proceedings of the IX International Workshop on Professional Retraining and Life-Long Learning using ICT: Person-oriented Approach (3L-Person 2024)* (Vol. 3781, pp. 10–29). CEUR Workshop Proceedings. <https://ceur-ws.org/Vol-3781/paper01.pdf>
- Malik, S. I., Tawafak, R. M., & Shakir, M. (2021). Aligning and assessing teaching approaches with SOLO taxonomy in a computer programming course. *International Journal of Information and Communication Technology Education*, 17(4), 1–15. <https://doi.org/10.4018/IJICTE.20211001.0a5>
- Pan, M., Lin, S., & Xiao, Z. (2025). From code analysis to fault localization: A survey of graph neural network applications in software engineering. *International Journal of Advanced Computer Science and Applications*, 16(4), 609–617. <https://doi.org/10.14569/IJACSA.2025.0160461>
- Piech, C., Spencer, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L., & Sohl-Dickstein, J. (2015). Deep knowledge tracing. *Advances in Neural Information Processing Systems*, 28, 505–513. <https://doi.org/10.48550/arXiv.1506.05908>
- Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for CS assessment. In S. Hamilton & M. Hamilton (Eds.), *Proceedings of the tenth conference on Australasian computing education* (Vol. 78, pp. 155–161). Australian Computer Society. <https://dl.acm.org/doi/10.5555/1379249.1379265>
- Wang, J., Li, Y., & Zhang, T. (2021). Process-based AI assessment framework for programming. *Computers and Education: Artificial Intelligence*, 2, 100045.
- Whalley, J. L., Clear, T., Robbins, P., & Thompson, E. (2011). Salient elements in novice solutions to code writing problems. In *Proceedings of the 13th Australasian Computing Education Conference (ACE 2011)* (pp. 99–107). Australian Computer Society.