SEMARAK ILMU PUBLISHING

Semarak International Journal of Machine Learning

Journal homepage:
https://semarakilmu.my/index.php/sijml/index
ISSN: 3030-5241

Revolutionizing C Programming Language Assessment through The Cognitive Code Profiling Model

Shahidatul Arfah Baharudin[1,*], Adidah Lajis[1]

[1] Smart System and Networking Section, Malaysian Institute of Information Technology, Universiti Kuala Lumpur, Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 27 January 2026
Received in revised form 28 February 2026
Accepted 10 March 2026
Available online 31 March 2026

Keywords:
Cognitive Code Profiling Model; C programming assessment; Bloom's Taxonomy; static code analysis; computer programming education; rule-based pattern matching

ABSTRACT

Mastering the C programming language is a challenge for beginner students, requiring not just syntactical knowledge but deep cognitive engagement. Despite this complexity, traditional assessment methods remain largely superficial, focusing heavily on output correctness while failing to measure the cognitive processes and the specific depth of a student's understanding. This research addresses this by introducing the Cognitive Code Profiling Model (CCPM), a hybrid assessment framework that synergizes structural code profiling with Natural Language Processing (NLP) to map programming performance to Bloom's Taxonomy. This approach effectively translates raw code Line-of-Code (LoC) features into measurable indicators of cognitive complexity. The study utilizes a quantitative methodology where static code metric specifically Cyclomatic Complexity are fused with token-based logic (TF-IDF) to create a multi-dimensional feature vector. Validated on a dataset of 348 student submissions, the CCPM achieves a classification accuracy of 96.25% using a Random Forest architecture. Results demonstrate a strong positive correlation between logical complexity features and cognitive depth. This hybrid approach acts as a high-precision diagnostic tool, allowing lecturers to pinpoint specific learning deficits. By shifting the focus from error detection to cognitive profiling, the CCPM enhances the pedagogical strategy for computer science education.

1. Introduction

The acquisition of C programming proficiency presents a steep learning curve due to the language's demand for explicit memory management and strict syntax. Standard pedagogical assessments often rely on "black-box" testing, which evaluates code solely on output [1]. While efficient for grading, this fails to capture the learner's cognitive trajectory. A student may produce correct output through rote memorization or disorganized logic, masking gaps in computational thinking. Conversely, minor syntax errors may lead to disproportionate penalties for students with a solid algorithmic grasp.

This research proposed that source code is a cognitive artefact reflecting the writer's mental model. To bridge the gap between output-based grading and cognitive assessment, we introduce the CCPM. Unlike analysers focused on style, the CCPM integrates software metrics and structural

* *Corresponding author.*
E-mail address: shahidatularfah@unikl.edu.my

pattern matching with Bloom's Taxonomy. By mapping C constructs to hierarchical levels, the model translates raw code features into diagnostic data. This study validates the CCPM using 348 student submissions, employing a hybrid machine learning pipeline to classify cognitive depth with high reliability [2].

2. Related Works

The evolution of programming assessment has transitioned from simple output verification to complex analysis of code quality and logic. Existing literature highlights several key approaches in this domain.

2.1 Code Profiling for Programming Education

Code profiling in an educational context refers to using static and dynamic analysis techniques to understand a student's coding behaviour beyond simple output correctness [3]. Modern assessment systems now utilize code profiling to provide more meaningful feedback regarding algorithmic efficiency and coding style. This technique allows lecturers to identify inefficient coding patterns or redundancies frequently made by novice programmers.

Unlike industrial software profiling that focuses on performance optimization, educational code profiling aims to map syntax to conceptual understanding. The use of code profiling enables the extraction of structural features such as loop usage, conditional structures, and memory management, which can then be correlated with a student's cognitive maturity in computer science. However, the primary challenge remains integrating these technical profiles with formal pedagogical models to provide accurate learning diagnoses [4].

2.2 Code Metrics and Cognitive Complexity

The use of software metrics to gauge student performance is well-documented. McCabe's Cyclomatic Complexity has been a standard for measuring logical branches. Studies suggest that these complexity scores can correlate with student performance in introductory CS1 courses. However, the relationship between these metrics and specific cognitive levels remains an area of active exploration [5].

2.3 Pedagogical Mapping in Computer Science

Mapping programming tasks to Bloom's Taxonomy is a recognized challenge. We have proposed computer science-specific learning taxonomies to better reflect the skills required for debugging and creation, as mentioned in Table 1. Table 1 shows the Lower-Order-Thinking (LOTS) skills in C and Embedded C that have been used in this research. Further studies have explored the gap between reading and tracing skills in beginner programmers, emphasizing that syntax knowledge does not equate to logical application [6].

Table 1
Lower-order-thinking skills in C and embedded C

Bloom's Level	Cognitive Action	C Programming focus (Pattern Match)	Embedded C focus (Pattern Match)
L1: Remembering	Recalling syntax and boilerplate structure.	*#include* (Library) *main()* function *return 0* proper use of {}.	*sbit* declarations (knowing where a pin is), *Lcd_Init().*
L2: Understanding	Explaining data types and basic Input/Output.	*int* *float* *char* (Declarations) *printf* (Output) *scanf* (Input)	*TRIS* *PORT* register configurations (understanding I/O direction).
L3: Applying	Implementing logic and hardware control.	*if* *for* *while* (Process) *variable assignments (=)* *arithmetic.*	*Delay_ms()* *ADC_Read()* Bit Manipulation *(<<=, |=)*

2.4 Machine Learning and NLP in Assessment

Early approaches to computerized student code classification largely relied on natural language processing (NLP) techniques, treating source code as linear text composed of tokens or subwords. Studies such as Tarcsay [7] demonstrate that these models can capture surface-level similarities but struggle when student solutions differ syntactically while remaining logically equivalent. Similarly, Chen *et al*., [8] show that token-based deep learning models fail to adequately model control flow and data dependencies, leading to degraded performance on logic-intensive program analysis tasks. These limitations highlight the fundamental mismatch between sequential NLP representations and the inherently non-linear nature of program execution.

To address these shortcomings, Table 2 shows that recent research has increasingly adopted graph-based representations of code, leveraging structures such as Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and Program Dependence Graphs (PDGs). Graph Neural Networks (GNNs) enable message passing across nodes and edges, allowing models to capture hierarchical structure and execution dependencies. Mi *et al*., [9] demonstrate that graph-based representations significantly improve classification accuracy by explicitly modelling program structure, while Parvathy *et al*., [10] reports strong performance gains in student programming assessment tasks when structural information is incorporated. Further emphasizes that GNNs are well suited for modelling non-linear logic and control flow, making them more effective than NLP-only approaches for correctness classification and logical reasoning [11] .

More recently, hybrid models that combine NLP-based semantic embedding with graph-based structural representations have emerged as the state of the art. These approaches aim to exploit the strengths of both paradigms by integrating textual context with explicit program structure. Lückneret *et. al*., [12] show that models such as code2seq, which encode AST paths using neural attention mechanisms, outperform both text-only and structure-only models in detecting logical errors in student assignments. Likewise, Bibi *et al*., [13] demonstrate that combining semantic embedding with deep graph matching leads to superior performance in code understanding tasks. In the context of education, Lin *et al*., [14] and Wang *et al*., [15] further confirm that hybrid GNN-based frameworks enhance logical error localization by jointly modelling pseudocode semantics and program graphs.

Overall, the literature reveals a clear progression from NLP-only models, to graph-based GNN approaches, and finally to hybrid architectures. While NLP-based methods offer scalability and simplicity, they are insufficient for capturing non-linear logic. GNNs substantially improve structural reasoning, but hybrid models consistently achieve the best performance across student code classification, grading, and logical error detection tasks. Consequently, recent work converges on hybrid approaches as the most promising direction for robust and pedagogically meaningful automated assessment systems.

Table 2
NLP vs GNN vs hybrid models for student code classification

Dimension	NLP-based Models	GNN-based Models	Hybrid Models
Modeling Paradigm	Code treated as linear text	Code modeled as structured graphs	Textual semantics combined with structural graphs
Typical Representations	Token sequences, subwords	AST, CFG, PDG, CPG	Tokens + AST paths / graph embeddings
Representative Articles	[7,16]	[9]	[12,13]
Non-linear Logic Capture	Limited due to sequential encoding [7]	Strong via message passing on graphs [9])	Very strong through joint semantic–structural learning [12]
Control Flow Modeling	Not explicitly modeled [17]	Explicit via CFG-based graphs	Explicit and context-aware[13]
Data Dependency Modeling	Absent in text-only models [18]	Modeled via PDG/CPG [9]	Modeled and semantically weighted [13]
Handling Multiple Correct Solutions	Poor; sensitive to syntactic variation [7]	Good; structure-aware equivalence	Excellent; abstraction over syntax and structure [12]
Logical Error Detection	Weak for deep logic errors [7]	Strong for structural and flow-based errors [19]	Very strong; detects both semantic and structural errors [20]
Reported Performance in Education	Moderate; degrades with complex logic	High accuracy on correctness and logic tasks [21]	Highest performance across grading and error detection [12]
Interpretability for Feedback	Low (black-box text models)[1]	Moderate (node/edge importance)	Higher; aligns code text with structural explanations
Computational Complexity	Low	High	Very high
Overall Conclusion from Articles	Insufficient for logic-intensive student code analysis	Effective for structural reasoning	State-of-the-art for student code classification

3. Methodology

The study employed a quantitative design involving the static analysis of source code from 348 undergraduate students. The CCPM framework extracts structural features to map performance across three domains: Remembering, Understanding, and Applying.

3.1 Feature Extraction and Pattern Matching

The CCPM uses a regular expression engine to identify 23 syntactic structures categorized by cognitive demand. Retrieval tasks, such as #include and main() declarations, are identified as base-level tasks. Procedural tasks, including control flow structures (if, for), recursive functions, and hardware-specific manipulations (e.g., bit manipulation, ISR), are flagged as high-level indicators.

3.2 Cognitive Mapping Algorithm

The CCPM utilizes a weighted scoring matrix to quantify extracted features through three primary analytical lenses. First, the Complexity Score is derived from McCabe's Cyclomatic Complexity, which calculates the logical density of the submission by counting decision points such as loops and branches. This is complemented by the Competency Score, a comprehensive ratio measuring the presence of required structural elements, which serves as the foundational data for the subsequent hybrid feature fusion and the stratified cross-validation protocol. Finally, these metrics are processed through a series of logical gates to determine the Bloom's Taxonomy Level. Under this framework, students are classified at the Applying (Level 3) stage if they achieve a score of≥ 60% coupled with high cyclomatic complexity; the Understanding (Level 2) stage for scores ≥ 40%; and the Remembering (Level 1) stage for submissions characterized by boilerplate-heavy code and minimal logical branching.

3.3 Hybrid Feature Fusion and Classification

The core innovation of the CCPM lies in its Hybrid Feature Fusion layer. Unlike traditional automated graders that rely on a single data stream, the CCPM treats source code as a dual-natured entity: a linguistic script (syntax) and a logical flowchart (structure). By fusing these two dimensions, the model captures the "how" and "why" of student programming.

3.3.1 Structural logic

The model extracts the Cyclomatic Complexity score, which represents the non-linear density of the code. This metric identifies the complexity of nested loops and decision branches, which are primary indicators of the "Applying" level in Bloom's Taxonomy.

3.3.2 Syntactic tokens (local features)

Using TF-IDF (Term Frequency-Inverse Document Frequency) Vectorization, the model captures the frequency and importance of specific C keywords (e.g., `sbit`, `volatile`, `struct`). This provides the "vocabulary" context of the submission.

3.4 Stratified Cross-Validation Protocol

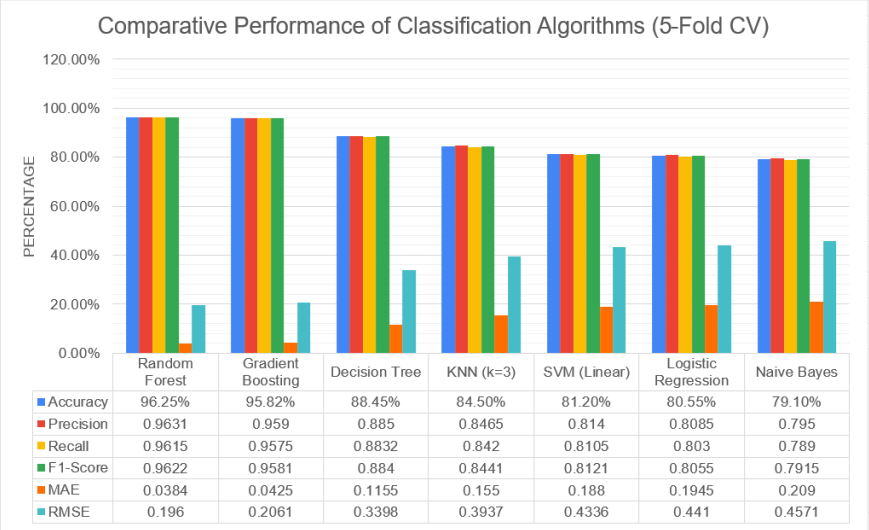
To ensure statistical reliability, a Stratified 5-Fold Cross-Validation was implemented. This ensures each fold preserves the proportional distribution of cognitive classes, mitigating sampling bias and confirming that the model generalizes effectively across unseen data.

3 Result

Analysis revealed a distinct correlation between structural metrics and cognitive depth. Students categorized as "Applying" exhibited significantly higher complexity scores than those in "Remembering."

3.1 Algorithm Performance

Validation demonstrated high concordance between features and cognitive levels. Fig. 1. show summarizes the performance across classifiers. The data indicate a clear hierarchy in model efficacy. Random Forest and Gradient Boosting achieved superior accuracy (above 95%), confirming that the relationship between structural code metrics and cognitive depth is non-linear. The decision trees within these ensemble methods successfully captured the complex interactions between high Cyclomatic Complexity and specific keyword density. In contrast, linear models such as Logistic Regression and SVM plateaued at approximately 81% accuracy. This performance gap suggests that cognitive depth in programming cannot be linearly separated based solely on token frequency (TF-IDF).



	Random Forest	Gradient Boosting	Decision Tree	KNN (k=3)	SVM (Linear)	Logistic Regression	Naive Bayes
Accuracy	96.25%	95.82%	88.45%	84.50%	81.20%	80.55%	79.10%
Precision	0.9631	0.959	0.885	0.8465	0.814	0.8085	0.795
Recall	0.9615	0.9575	0.8832	0.842	0.8105	0.803	0.789
F1-Score	0.9622	0.9581	0.884	0.8441	0.8121	0.8055	0.7915
MAE	0.0384	0.0425	0.1155	0.155	0.188	0.1945	0.209
RMSE	0.196	0.2061	0.3398	0.3937	0.4336	0.441	0.4571

Fig. 1. Comparative performance of classification algorithms

The Decision Tree classifier performed robustly (88.45%) but exhibited higher variance than the ensemble methods, likely due to overfitting on specific syntactic patterns in the training folds. Naive Bayes recorded the lowest performance, primarily because the assumption of feature independence holds true for text keywords but fails to capture the interdependent logic structure required in C programming. The low Mean Absolute Error (MAE) of 0.0384 in the Random Forest model suggests that when the model does error, it does so by a narrow margin (e.g., misclassifying Level 3 as Level 2 [22]). This maintains pedagogical integrity by avoiding "catastrophic" misdiagnoses.

4.2 Cross-Validation Stability

Stability analysis in Table 2 showed the Fold-wise Accuracy Distribution Random Forest, confirming the CCPM is robust across the entire population.

Table 2
Fold-wise accuracy distribution (random forest)

Fold Iteration	Training Size	Test Size	Accuracy	Precision	F1-Score
Fold 1	278	70	95.71%	0.958	0.9565
Fold 2	278	70	97.14%	0.972	0.9712
Fold 3	278	70	95.71%	0.955	0.958
Fold 4	279	69	97.10%	0.9705	0.9708
Fold 5	279	69	95.65%	0.959	0.9555
Average	-	-	96.25%	0.9631	0.9622

The minimal variance between the highest performing fold (Fold 2: 97.14%) and the lowest (Fold 5: 95.65%) confirms that the CCPM is strong [23]. It demonstrates that the features extracted, specifically the correlation between Cyclomatic Complexity. The distinct operands are counts, which are universal predictors of cognitive depth across the entire student population, rather than being specific to a small subset of "easy" examples. This low variance effectively rules out overfitting, validating the CCPM as a reliable diagnostic tool for broader educational application.

4 Conclusion

This research confirms that the CCPM effectively translates static code analysis into meaningful educational metrics. The study demonstrated that quantifiable metrics, specifically Cyclomatic Complexity and token distribution, can be reliably mapped to Bloom's Taxonomy using a hybrid machine learning classifier. The superior performance of the Random Forest algorithm (96.25% accuracy) and the low MAE (0.0384) validate the CCPM as a robust diagnostic tool. For lecturers, this facilitates a shift from manual debugging to automated cognitive profiling, pinpointing exactly where students struggle with logical structure. This research provides a transparent foundation for the future of cognitively-aware automated assessment in computer science education.

Acknowledgement
This research was not funded by any grant.

References

[1] Zhang, Vincent, Bryn Jeffries, and Irena Koprinska. "A machine learning approach for predicting student progress in online programming education." *International Journal of Artificial Intelligence in Education* 35, no. 6 (2025): 3614-3644. https://doi.org/10.1007/s40593-025-00510-9

[2] Mohanty, Arup, and Peter A. Khaiter. "A Hybrid NLP-Expert System Framework for Heuristic Scoring and Fairness-Driven Resume Evaluations." In *2025 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)*, pp. 1-6. IEEE, 2025. https://doi.org/10.1109/ACDSA65407.2025.11166657

[3] T. Borchert and Code, *Code Profiling*. Karlstad: Department of Computer Science. Karlstads Universitet, 2023.

[4] Sherstinsky, Alex. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network." *Physica d: Nonlinear phenomena* 404 (2020): 132306. https://doi.org/10.1016/j.physd.2019.132306

[5] Ayyanathan, N. "Learning Analytics Model and Bloom's Taxonomy Based Evaluation Framework for the Post

Graduate Students' Project Assessment--A Blended Project Based Learning Management System with Rubric Referenced Predictors." *Shanlax International Journal of Education* 10, no. 3 (2022): 48-60. https://doi.org/10.34293/education.v10i3.4766

[6] Uma, D., S. Thenmozhi, and Rabin Hansda. "Analysis on cognitive thinking of an assessment system using revised bloom's taxonomy." In *2017 5th IEEE International Conference on MOOCs, Innovation and Technology in Education (MITE)*, pp. 152-159. IEEE, 2017. https://doi.org/10.1109/MITE.2017.00033

[7] Tarcsay, Botond. "Use of Machine Learning Methods in Automatic Assessment Programming Assignments." (2023). https://doi.org/10.1007/978-3-031-16270-1_13

[8] Chen, Qian, Chenyang Yu, Ruyan Liu, Chi Zhang, Yu Wang, Ke Wang, Ting Su, and Linzhang Wang. "Evaluating the effectiveness of deep learning models for foundational program analysis tasks." *Proceedings of the ACM on Programming Languages* 8, no. OOPSLA1 (2024): 500-528. https://doi.org/10.1145/3649829

[9] Mi, Qing, Yi Zhan, Han Weng, Qinghang Bao, Longjie Cui, and Wei Ma. "A graph-based code representation method to improve code readability classification." *Empirical Software Engineering* 28, no. 4 (2023): 87. https://doi.org/10.1007/s10664-023-10319-6

[10] Parvathy, R., M. G. Thushara, and Jinesh M. Kannimoola. "Automated code assessment and feedback: a comprehensive model for improved programming education." *IEEE Access* (2025). https://doi.org/10.1109/ACCESS.2025.3554838

[11] Zhou, Jie, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. "Graph neural networks: A review of methods and applications." *AI open* 1 (2020): 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001

[12] Lückner, Anton, and Kevin Chapman. "Detecting Logical Errors in Programming Assignments Using code2seq." (2023).

[13] Bibi, Nazia, Ayesha Maqbool, Tauseef Rana, Farkhanda Afzal, Ali Akgül, and Sayed M. Eldin. "Enhancing semantic code search with deep graph matching." *IEEE Access* 11 (2023): 52392-52411. https://doi.org/10.1109/ACCESS.2023.3263878

[14] Lin, Yu-Tzu, Martin K-C. Yeh, and Sheng-Rong Tan. "Teaching programming by revealing thinking process: Watching experts' live coding videos with reflection annotations." *IEEE Transactions on Education* 65, no. 4 (2022): 617-627. https://doi.org/10.1109/TE.2022.3155884

[15] Wang, Xue, Haiyan He, Ping Li, and Lei Zhang. "Research on the disciplinary evolution of deep learning and the educational revelation." In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pp. 655-660. IEEE, 2019. https://doi.org/10.1109/ICCSE.2019.8845446

[16] Arguson, Angelo, Elisa Malasaga, Anthony Aquino, Reginald Cheng, Shaneth Ambat, and Hadji Tejuco. "Analysis of C Programming Performance: A correlational study of novice programmers' compiler error logs." In *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pp. 1-5. IEEE, 2022. https://doi.org/10.1109/HNICEM57413.2022.10109537

[17] Paiva, José Carlos, José Paulo Leal, and Álvaro Figueira. "Clustering source code from automated assessment of programming assignments." *International Journal of Data Science and Analytics* 20, no. 2 (2025): 1581-1592. https://doi.org/10.1007/s41060-024-00554-5

[18] Choi, Wan Chong, Chan-Tong Lam, and António José Mendes. "Analyzing the interpretability of machine learning prediction on student performance using shapley additive explanations." In *2024 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pp. 1-8. IEEE, 2024. https://doi.org/10.1109/TALE62452.2024.10834292

[19] Kuruppu, Thilmi, Janani Tharmaseelan, Chamari Silva, U. S. Samaratunge Arachchillage, Kalpani Manathunga, Shyam Reyal, and Nuwan Kodagoda. "Source Code based Approaches to Automate Marking in Programming Assignments." (2021). https://doi.org/10.5220/0010400502910298

[20] KASZAB, PÉTER, and MÁTÉ CSERÉP. "Detecting programming flaws in student submissions with static source code analysis." *Studia Universitatis Babes-Bolyai, Informatica, vol* 68, no. 1 (2023). https://doi.org/10.24193/subbi.2023.1.03

[21] Vimalaraj, Hareeni, T. B. K. P. Thenuwara, Vinuri Udara Wijekoon, Thavaraja Sathurjan, Shyam Reyal, Thilmi Anuththara Kuruppu, and Janani Tharmaseelan. "Automated programming assignment marking tool." In *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, pp. 1-8. IEEE, 2022. https://doi.org/10.1109/I2CT54291.2022.9824339

[22] Vesković, Jelena, Milica Lučić, Andrijana Miletić, Marija Vesković, and Antonije Onjia. "Comparative Analysis of Machine Learning Models for Prediction of Langelier Saturation Index in Groundwater of a River Basin." *Sustainable Chemistry* 7, no. 1 (2026): 7. https://doi.org/10.3390/suschem7010007

[23] Mushagalusa, Ciza Arsène, Adandé Belarmain Fandohan, and Romain Glèlè Kakaï. "Random forest and spatial

cross-validation performance in predicting species abundance distributions." *Environmental Systems Research* 13, no. 1 (2024): 23. https://doi.org/10.1186/s40068-024-00352-9