MyJICT
Malaysian Journal of Information and Communication Technology
EISSN 0127-7812

Vol. 10, Issue 2, pp. 48-54 | DECEMBER 2025
DOI: https://doi.org/10.53840/myjict10-2-226

A COGNITIVE LEVEL ASSESSMENT BASED ON BLOOM'S TAXONOMY VIA NLP AND LINE-BY-LINE METHODS: FOCUS ON LOW-LEVEL COGNITIVE COMPETENCY

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

[i]Computer Engineering Technology, Malaysian Institute of Information Technology, Universiti Kuala Lumpur

*(Corresponding author) e-mail: shahidatularfah@unikl.edu.my

Article history (leave this part):
Submission date: 14 October 2025
Received in revised form: 21 November 2025
Acceptance date: 22 November 2025
Available online: 17 December 2025

Keywords:
Bloom's Taxonomy, Cognitive Competency, Natural Language Processing, Educational Assessment, Programming

Funding:
This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Competing interest:
The author(s) have declared that no competing interests exist.

Cite as:
Baharudin, S. A., & Lajis, A. (2025). A Cognitive Level Assessment Based on Bloom's Taxonomy Via NLP and Line-By-Line Methods: Focus on Low-Level Cognitive Competency. Malaysian Journal of Information and Communication Technology (MyJICT), 10(2), 48-54. https://doi.org/10.53840/myjict10-2-226

ABSTRACT

This paper describes the evaluating cognitive skills is crucial for providing insights into a learner's intellectual competence and readiness to engage with various levels of knowledge. This work presents a new methodology for evaluating cognitive levels, focusing on low-level cognitive competencies, using Natural Language Processing (NLP) and line-by-line analysis. One of the most prominent educational frameworks, Bloom's Taxonomy, classifies cognitive skills into levels, ranging from basic recall to more advanced thinking. However, research has shown that traditional approaches often fail to measure lower-level cognitive processes like remembering or understanding accurately. This gap can be addressed by using NLP techniques to analyze textual feedback on a line-by-line basis. The proposed approach uses advanced NLP algorithms to identify line-by-line code C language programming assessments that align with the cognitive levels described in Bloom's Taxonomy. Each response line is examined to assess whether it corresponds to foundational cognitive skills like knowledge recall and comprehension. This detailed approach allows for more precise differentiation of various cognitive competencies within lower-order thinking. This methodology has been validated through analysis of a large dataset of student responses across multiple disciplines, with results compared to traditional methods. This research benefits the educational community by offering a scalable approach to cognitive assessment that can be integrated into e-learning platforms. In conclusion, this study demonstrates the potential of combining NLP methods with Bloom's Taxonomy to assess lower-level cognitive skills in a more effective, scalable, and precise way. The line-by-line analysis provides valuable insights into student understanding, supporting evidence-based educational evaluation.

Introduction

Assessing students' cognitive skills is a foundation of effective education. It provides lecturers with critical insights into a student's understanding of a subject and their readiness to do complex material. Among the most influential frameworks for categorizing these skills is Bloom's Taxonomy, which outlines a hierarchy of cognitive levels from foundational recall to refined creation (Anderson et al., 2001) . While invaluable, the practical application of this taxonomy, especially in large-scale educational settings, presents significant challenges. Particularly in university, subjects like computer programming, in traditional assessment methods often fall short. While auto-grading systems only focus on functional correctness, they assess whether the code runs and produces the correct output. Primarily evaluates the final product rather than the cognitive process employed by the student (Ala-Mutka, 2005). Manual grading by instructors, while more nuanced, is time-consuming and prone to inconsistency, making it impractical for large classes (Conejo et al., 2019). This creates an acute gap in assessment, especially at the lower levels of Bloom's Taxonomy: *Remembering*, *Understanding, and Applying*. These foundational skills, which involve recalling facts and explaining concepts, are the building blocks for all higher-order thinking but are often measured lightly.

To address this challenge, this paper proposes a methodology that uses Natural Language Processing (NLP) with a line-by-line analysis of C language programming code. By treating code as a form of structured text, we can apply NLP algorithms to dissect student submissions at a granular level (Somers et al., 2021). This approach allows us to map specific lines of code to the low-level cognitive competencies of *Remembering*, *Understanding and Applying* as defined by Bloom's Taxonomy. The goal is to create a more accurate model for cognitive assessment that provides deeper insights into the student learning process. This study provides a framework for integrating nuanced feedback through evidence-based evaluation, ultimately improving input for both students and lecturers.

Literature Review

Bloom's Taxonomy in Computer Science Education

The framework known as Bloom's Taxonomy, along with its revised iteration (Anderson et al., 2001), has been widely adopted within the domain of computer science education (CSE) for curriculum design, the formulation of learning objectives, and the development of assessments (Masapanta-Carrión & Velázquez-Iturbide, 2018). The hierarchical arrangement of the taxonomy, Remembering, Understanding, Applying, Analysing, Evaluating, Creating, offers a lucid framework for scaffolding educational experiences, transitioning from the basic recall of syntax to the intricate design of software systems. Scholars have employed this framework to categorize the cognitive demands inherent in examination queries and programming assignments (Smith et al., 2023). Nevertheless, a persistent challenge remains in the objective and consistent evaluation of these cognitive levels. A comprehensive review conducted by (Masapanta-Carrión & Velázquez-Iturbide, 2018) revealed that the most frequently encountered difficulty reported by instructors was the classification of tasks into specific cognitive levels. While higher-order competencies, such as Applying and Creating, are often manifest in a complete and functional program, the foundational competencies of Remembering and Understanding are intricately interwoven within the structure and syntax of the code, rendering them more challenging to isolate and assess reliably.

Automated Assessment of Programming Assignments

The necessity for proficient evaluation within programming courses has culminated in the creation of a plethora of automated assessment instruments. These instruments predominantly serve as autograders, compiling and executing student-generated code against a predetermined set of test cases (Ala-Mutka, 2005). While they are effective in ascertaining functional accuracy and providing instantaneous feedback, this methodology is not without its shortcomings. It frequently functions as a "black box," awarding

students for correct output irrespective of the quality, efficiency, or the student's foundational understanding of the code (Hart et al., 2023). It encounters difficulties in distinguishing between a student who comprehends a concept and one who has simply committed a code snippet to memory. Consequently, these systems offer minimal insight into the cognitive processes associated with the lower tiers of Bloom's taxonomy.

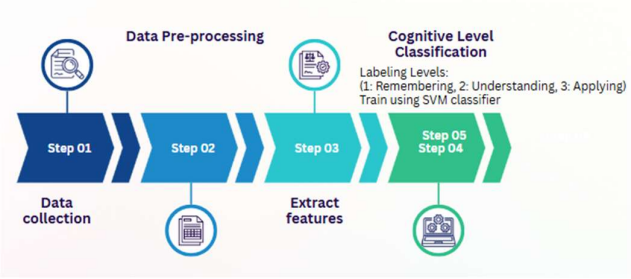
Natural Language Processing for Source Code Analysis

In recent years, the application of NLP techniques to source code has emerged as a promising field of research (Zhu et al., 2022). Viewing source code as a specialized language, researchers have applied NLP models to tasks such as code summarization, bug detection, and authorship attribution (Allamanis et al., 2019). This paradigm of "code as text" opens up new possibilities for educational assessment. Previous work has explored using NLP to analyze student self-explanations of code, demonstrating a link between the quality of explanation and comprehension (Chapagain et al., 2022). More advanced techniques now focus on associating natural language comments directly with code entities and adapting large pre-trained models for code-specific tasks (Mekterović et al., 2023). Our research extends this concept by applying NLP directly to the code itself, using a line-by-line analysis to infer the cognitive skills demonstrated in its construction, a method not yet extensively explored for direct mapping to Bloom's lower-level competencies.

Methodology

Our proposed methodology assesses the low-level cognitive skills in C programming assignments through a three-stage NLP-based framework is Data Pre-processing, Feature Extraction and Mapping, and Cognitive Level Classification.

Figure 1: Research Methodology



Data Pre-processing

The methodology was developed by using a dataset of 93 introductory C programming submissions collected from a first-year undergraduate course to get the initial findings. In the pre-processing stage, each C code file is first parsed to ensure syntactic validity. Comments and non-essential whitespace are removed. The core of this stage is the line-by-line tokenization, where each line of code is broken down into a sequence of tokens, including keywords, identifiers, operators, and literals as mention in Figure 2.

Figure 2: Process Documents using Rapid Miner

Feature Extraction and Mapping to Bloom's Taxonomy

This stage is the keystone of our approach. We developed a rule-based mapping system that associates specific code features, identifiable at the line level, with the two lowest levels of Bloom's Taxonomy: Remembering, Understanding and Applying. This mapping is based on the cognitive actions required to write a given line of code(Nguyen et al., 2020). To process the documents, the sample must follow the logic structure classifications. There are Library, Main Function, and Declaration. Input, Process, Output, return 0, and Code Block. Each of the logic structure will be assign in Bloom's Taxonomy as below:

a) Remembering: Library, Output, return 0
b) Understanding: Main Function, Declaration
c) Applying: Looping, Selection

A classification model, utilizing a Support Vector Machine (SVM) algorithm, was trained on these features. Each line of code is converted into a feature vector representing the presence or absence of these predefined code patterns, a process informed by techniques like enhanced TF-IDF used in similar classification tasks (Mohammedid & Omar, 2020).

Cognitive Level Classification

The methodology employs a trained classification model to identify cognitive assessment. This model takes a student's submission, converts each line into a feature vector, and classifies it into a defined cognitive category *'Remembering', 'Understanding', Applying* '. The output is an annotated version of the student's code, where each line is tagged with its assessed cognitive level as mention in Table 1. This approach aligns with other efforts to automate the classification of educational materials based on Bloom's Taxonomy using machine learning. This provides a granular profile of the student's work, highlighting their strengths and potential areas of misunderstanding.

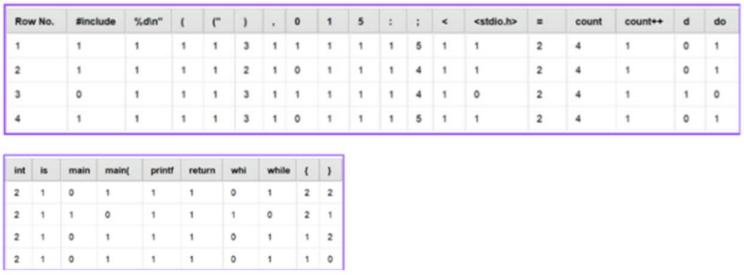
Table 1: Mapping C Code Constructs to Low-Level Bloom's Taxonomy

C Code Line Example	Cognitive Level	Justification
#include <stdio.h>	**Remembering**	Recalling the exact syntax and name of a standard library.
int student_age;	**Remembering**	Recalling the syntax for declaring a variable of a specific type.
printf("Hello");	**Remembering**	Recalling the name and basic usage of a standard output function.
average = sum / count;	**Understanding**	Demonstrating comprehension of arithmetic operations and assignment.
if (score >= 60)	**Understanding**	Demonstrating comprehension of conditional logic and relational operators.
for (i=0; i<10; i++)	**Understanding**	Demonstrating comprehension of iterative control flow structures.

Results and Findings

As in Figure 3, Rows 2, 3, and 4 indicate the student's answer. Row 1 indicates the lecturer's answer. If one of the attributes did not appear in the same answers as Row 1, it indicates that the student made an error when answering the assessment. This is part of a student's answer.

Figure 3: Results using Term Occurrences



Row No.	#include	%d\n"	(	("	)	,	0	1	5	:	;	<	<stdio.h>	=	count	count++	d	do
1	1	1	1	1	3	1	1	1	1	1	5	1	1	2	4	1	0	1
2	1	1	1	1	2	1	0	1	1	1	4	1	1	2	4	1	0	1
3	0	1	1	1	3	1	1	1	1	1	4	1	0	2	4	1	1	0
4	1	1	1	1	3	1	0	1	1	1	5	1	1	2	4	1	0	1

int	is	main	main(	printf	return	whi	while	{	}
2	1	0	1	1	1	0	1	2	2
2	1	1	0	1	1	1	0	2	1
2	1	0	1	1	1	0	1	1	2
2	1	0	1	1	1	0	1	1	0

The analysis of student performance reveals three distinct profiles of programming errors, all identified through token mismatches against the lecturer's correct answer (Row 1). Student 3 (Row 3) made the most severe structural errors, indicated by the complete omission of the necessary input/output header (#include and <stdio.h> counts of 0) and the main function definition (count of 0), resulting in non-compilable code. Student 4 (Row 4) displayed fundamental logic and operator errors, failing to include the required iterative control flow by omitting both the do and while keywords (count of 0) and instead overusing operators like = (3 extra) and . (2 extra), showing a deep misunderstanding of the problem's logic. In contrast, Student 2 (Row 2) made only minor structural errors, such as one fewer) symbol and one fewer 0 literal, suggesting that their solution was syntactically close to the correct answer but contained small implementation flaws.

Validation and Precision

To validate the model's accuracy, a random subset of 93 submissions was manually annotated line-by-line by a panel of one computer science lecturer. The panel consensus was used as the ground truth. The validation of scoring systems is a critical step to ensure their reliability and fairness. The validation phase will be proceed on the next phase.

Conclusion

This study introduces a novel methodology for assessing foundational cognitive skills in programming. By innovatively combining a line-by-line source code analysis with Natural Language Processing (NLP) techniques, the system effectively addresses the gap in accurately measuring lower-level cognitive processes. Specifically, the approach maps C language programming constructs to the Remembering, Understanding, and Applying levels of Bloom's Taxonomy. The resulting NLP-based framework offers a more effective, precise, and scalable alternative to traditional manual grading, particularly in large classes. This method is significant because it provides a granular profile of a student's work, enabling lecturers to determine a student's cognitive level at an early stage and deliver thorough, individualized feedback. Ultimately, this research supports evidence-based educational evaluation by providing valuable insights into student learning patterns for early, focused support.

Acknowledgement

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, *15*(2), 83–102. https://doi.org/10.1080/08993400500150747

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2019). A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, *51*(4). https://doi.org/10.1145/3212695

Anderson, L. W., Krathwohl Peter W Airasian, D. R., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J., & Wittrock, M. C. (2001). *Taxonomy for_ Assessing a Revision 0F Bl00M'S Tax0N0My 0F Educati0Nal Objectives*. https://www.uky.edu/~rsand1/china2018/texts/Anderson-Krathwohl - A taxonomy for learning teaching and assessing.pdf

Chapagain, J., Tamang, L., Banjade, R., Oli, P., & Rus, V. (2022). Automated Assessment of Student Self-explanation During Source Code Comprehension. *Proceedings of the International Florida Artificial Intelligence Research Society Conference, FLAIRS*, *35*. https://doi.org/10.32473/flairs.v35i.130540

Conejo, R., Barros, B., & Bertoa, M. F. (2019). Automated Assessment of Complex Programming Tasks Using SIETTE. *IEEE Transactions on Learning Technologies*, *12*(4), 470–484. https://doi.org/10.1109/TLT.2018.2876249

Hart, R., Hays, B., McMillin, C., Rezig, E. K., Rodriguez-Rivera, G., & Turkstra, J. A. (2023). Eastwood-Tidy: C Linting for Automated Code Style Assessment in Programming Courses. *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, *1*, 799–805. https://doi.org/10.1145/3545945.3569817

Hwang, W.-Y., Shadiev, R., Wang, C.-Y., & Huang, Z.-H. (2012). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. *Computers & Education*, *58*(4), 1267–1281. https://doi.org/10.1016/j.compedu.2011.12.009

K. Dalal, M., & A. Zaveri, M. (2011). Automatic Text Classification: A Technical Review. *International Journal of Computer Applications*, *28*(2), 37–40. https://doi.org/10.5120/3358-4633

Masapanta-Carrión, S., & Velázquez-Iturbide, J. Á. (2018). A systematic review of the use of Bloom's taxonomy in computer science education. *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, *2018-Janua*, 441–446. https://doi.org/10.1145/3159450.3159491

Mayer, R. E. (2002). A taxonomy for computer-based assessment of problem solving. *Computers in Human Behavior*, *18*(6), 623–632. https://doi.org/10.1016/S0747-5632(02)00020-1

Mekterović, I., Brkić, L., & Horvat, M. (2023). Scaling Automated Programming Assessment Systems. *Electronics (Switzerland)*, *12*(4). https://doi.org/10.3390/electronics12040942

Mohammedid, M., & Omar, N. (2020). Question classification based on Bloom's taxonomy cognitive domain using modified TF-IDF and word2vec. *PLoS ONE*, *15*(3), 1–21. https://doi.org/10.1371/journal.pone.0230442

Nguyen, P. H., Tangworakitthaworn, P., & Gilbert, L. (2020). Individual learning effectiveness based on cognitive taxonomies and constructive Alignment. *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, *2020-Novem*, 1002–1006. https://doi.org/10.1109/TENCON50793.2020.9293733

Panthaplackel, S., Gligoric, M., Mooney, R. J., & Li, J. J. (2020). Associating natural language comment and source code entities. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 8592–8599. https://doi.org/10.1609/aaai.v34i05.6382

Smith, D. H., Emeka, C., Fowler, M., West, M., & Zilles, C. (2023). Investigating the Effects of Testing Frequency on Programming Performance and Students' Behavior. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (Vol. 1, Issue 1). Association for Computing Machinery. https://doi.org/10.1145/3545945.3569821

Somers, R., Cunningham-nelson, S., & Boles, W. (2021). *Applying natural language processing to automatically assess student conceptual understanding from textual responses*. *37*(5), 98–115.

Yulianto, S. V., & Liem, I. (2014). Automatic grader for programming assignment using source code analyzer. *Proceedings of 2014 International Conference on Data and Software Engineering, ICODSE 2014*, 1–4. https://doi.org/10.1109/ICODSE.2014.7062687

Zhu, Q., Luo, X., Liu, F., Gao, C., & Che, W. (2022). A Survey on Natural Language Processing for Programming. *ArXiv*.