Journal of Applied Informatics and Computing (JAIC)
Vol.9, No.6, December 2025, pp. 2950~2956
e-ISSN: 2548-6861

Abstract Syntax Tree Model for Minimizing False Negative in Semantic Evaluation of Python Fill-in-the-Blank

Usman Nurhasan[1]*, Didik Dwi Prasetya[2], Syaad Patmanthara[3],
* Departemen Teknik Elektro dan Informatika, Universitas Negeri Malang, Indonesia
usman.nurhasan.2505349@students.um.ac.id[1], didikdwi@um.ac.id[2], syaad.ft@um.ac.id[3]

Article Info
Article history:
Received 2025-09-08
Revised 2025-11-04
Accepted 2025-11-08

Keyword:
Abstract Syntax Tree,
Epistemology of Evaluation,
Fill-in-the-Blank,
Pedagogical Effectiveness,
Semantic Evaluation,

ABSTRACT

This study develops and evaluates an automated assessment model using Abstract Syntax Trees (AST) with a view to overcoming the limitations of string-matching techniques in the assessment of Fill-in-the-Blank (FIB) programming answers. Traditional string-matching techniques have a relatively high False Negative Rate (FNR) of 21.5% within the context of detecting semantic equivalence. The current model uses semantic structural triangulation to ascertain the semantic similarity of student answers. Technical assessment shows that the AST approach markedly reduces the FNR to 4.5%. The model demonstrates high reliability ($\varkappa$ = 0.83) with high classification accuracy (F1 Score = 0.966) which attests to its inferential validity. From a pedagogical perspective, system implementation leads to substantial learning gains, evidenced by a large effect size (Cohen's d = 1.82) and a high normalized gain (Normalized Gain = 0.90). Multiple regression analysis confirms that semantic accuracy is the primary causal factor driving improved student comprehension. Ontologically, while AST is valid as a partial representation, its limitations—particularly tree isomorphism in recursive structures—highlight the need for further exploration of graph isomorphism approaches. Control Flow Graphs (CFG) and Data Flow Graphs (DFG) offer more expressive relational models for capturing control and data dependencies. The model demonstrates functional feasibility with a System Usability Scale (SUS) score of 76.47. Overall, the AST Triangulation Model is validated as pedagogically effective, inferentially robust, and supportive of evaluative transparency. Future research recommends validating the model on more complex tasks and releasing it as open-source to support reproducibility.

I. INTRODUCTION

Studies have shown that Python is primarily used across the world in CS1 and CS2 curricula where the teaching of foundational computational thinking is also integrated. The first two levels of the curriculum, for example, CS1, are essential for the use of automated assessment systems. CS1 tends to create the most false negative issues, specifically in the Fill-in-the-Blank (FIB) programming assignments. The students' responses are usually semantically equivalent but are presented in radically different syntactic forms that the string-matching systems fail to identify as correct.

In the CS2 curriculum, more complex topics are introduced, such as data structures and algorithms, which challenge the representational limits of pure Abstract Syntax Tree (AST) models. The recursive logic and complex data flows are significant problems that the AST-based systems have. It becomes necessary to shift to graph isomorphism, especially Control Flow Graphs (CFG) and Data Flow Graphs (DFG) to accurately capture control flows and semantic equivalence in the students' algorithms at CS2.

Agarwal showed that Python is the main language used for introductory programming courses for over 70% of the world's top universities [1]. This extensive use has Python being incorporated into automated assessment tools and other digital learning tools. Nevertheless, most of these tools that automate assessment focus on string matching techniques that do literal, line-by-line comparisons of code, ignoring the meaning and the logic of execution[2], [3]. Cheang has shown

that for elementary programming tasks, including FIB-type questions, asymmetry in the code can lead to a 28% false negative rate [4]. This means that the system will mark a logically true answer incorrect because the system will focus on syntactic differences.

In Indonesia, the national digital training programme as well as information technology programmes in Python have digital training programmes incorporated[5]. Prasetyo et al. states that above 60% of tertiary educational institutions in Indonesia integrated Python into LMS assessments [6]. However, these systems lack in being only text matching systems. This leads to frustration and a sense of inequity in assessment because of unconsidered semantic differences in student answers.

FIB activities are used to evaluate students' grasp of syntax and fundamental logic. However, some poorly designed automatic scoring systems misclassify answers that are functionally equivalent. Consider two function definitions that count the even numbers in a list – they may be different in syntax but compute the same result.

```
# Answer 1
def count_even(nums):
    return len([x for x in nums
if x % 2 == 0])
```

```
# Answer 2
def count_even(nums):
    count = 0
    for n in nums:
        if n % 2 == 0:
            count += 1
    return count
```

Even though the systems that use string matching fail to see the equivalence, both definitions produce the same output for all valid inputs. This leads to unfair scoring. In contrast, an Abstract Syntax Tree (AST) approach would permit detection of the equivalent definitions and thus achieve a definition of fair scoring [7]. Despite AST's considerable use in areas such as plagiarism detection and evaluation of semantic similarity, direct use of this technique to automate assessment of FIB programming tasks remains unexplored [8].

Notwithstanding its structural benefits, there remain studies without a semantic assessment framework using AST, particularly geared towards scenarios involving FIB. No other work leverages AST as a lightweight, efficient structural filter in automated assessment systems. Moreover, there is still no fully developed approach that pairs AST with execution results and expert critiques alongside one another, in a coherent fashion, to demonstrate semantic equivalence across student work [9].

This paper outlines a Lightweight and Valid Semantic Triangulation Model, which is directed towards automating FIB assessments within the field of Python training. The model in this instance looks to AST as a primary structural filter with execution results and expert annotations still retained as a filter. Validation is conducted through means of three methods: equivalent output comparison, inter-rater agreement (Cohen's Kappa > 0.8), and AST structural likeness evaluated by means of tree isomorphism. The intention is to curb false negatives, enhance precision in scoring, and uphold the perceived fairness of automated systems from the students' perspective [10].

Ontologically, the research argues that each programme has an abstract syntactic structure that captures its logic. The AST is treated as a partial semantic representation that captures some logical relations underlying the surface syntax, yet it falls short concerning some aspects of dynamism, such as the consequences of execution, runtime interaction, and execution behaviours[11]. Therefore, the AST serves as an approximation model, the validity of which is demonstrated through execution traces and expert evaluations.

The study is anchored epistemologically in evaluative-experimental design and post-positivism. Scientific truth is not an absolute construct but a product of triangulating empirical data from several sources. The integration of programme execution results, expert annotations, and AST analysis, in which semantic equivalence is sustained, enables validation. The three, in unison, constitute an epistemic triangulation approach that isolates the model's internal validity and provides an empirical comparison of its efficacy against traditional string matching methods.

II. METHOD

This study aims to assess the technical validity and the pedagogical value of a semantic assessment system based on Abstract Syntax Tree (AST) for Fill-in-the-Blank (FIB) Python programming activities. The methodology involves two main tracks. The first is the development and validation of the FIB assessment system based on a Research and Development (R&D) model. The second is the assessment of the FIB learning outcomes using a quasi-experimental design between groups.

A. Philosophical Foundations of FIB Activities

Fill-in-the-Blank (FIB) activities require students to complete programming tasks by filling in the gaps with code that shows an understanding of the logic of the programme. From an ontological approach, FIB responses are treated as abstract semantic entities with an Abstract Syntax Tree (AST) representation. The AST is a partial representation of the programme logic that can be assessed for empirical validity through structural equivalence and execution assessment [12]. With the adoption of structural realism, the system is assumed to capture the programme logic in an objective manner as long as the semantic interplay between the system components is preserved, even when their stylistic representations vary.

In identifying the epistemological stance, the method is defined as hybrid. The system's inferences on FIB responses are examined alongside quantitative measures encompassing accuracy, precision, and F1-score, which are also validated by experts with significant inter-rater reliability. Data triangulation is employed to guard the inferential validity and to build social trust in the automated evaluation system [13].

B. Research Design for FIB Evaluation

This study employs an R&D framework with iterative prototyping and integrates a quasi-experimental design to evaluate instruction on the constructed Fill-in-the-Blank (FIB) assessment system. These two methodologies are designed to run together and improve each other. The technical validation's purpose is to measure the performance of the AST Evaluator as a semantic classifier for FIB responses, which will utilise three different evaluation techniques. The first technique is accuracy analysis, where the performance of the system is evaluated against a truth benchmark defined by experts using precision, recall, F1-score, and the false negative rate to define ground truth [14]. The second technique is reliability validation, where expert annotations are evaluated using Cohen's Kappa alongside the system-generated scores and the scores of the experts. Third, qualitative error analysis informs iterative design, especially in the recursive and complex lambda functions [15].

The purpose of the pedagogical evaluation is to identify how the FIB-AST system functions in enhancing the learning outcomes and retention rates of students. The experimental design consists of one experimental group (n=26) which uses the FIB-AST system which generates automated semantic feedback, and one control group (n=25) which uses conventional teaching strategies. The same instructional materials and FIB tasks were given to both groups.

The assessments are given in three phases: Pre-Test, Post-Test, and Delay-Test. The Delay-Test is intended to measure transfer learning. Here, students determine semantic equivalence within new FIB tasks and previously learned knowledge. The Delay-Test items are structurally new and do not repeat previous questions. They present new situations that are logically the same, although differing in wording [16]. The validity of the test instruments was ensured by subject-matter experts while reliability was evaluated using Cronbach's alpha.

The analysis of data typically involves various quantitative methods. An Independent Sample T-Test is conducted to assess group differences in performance, and a Paired Sample T-Test is used to analyse score differences in the experimental group. To determine intervention effects, Cohen's d is used in conjunction with Normalised Gain, which assesses improvement relative to baseline scores. Furthermore, the conceptual variability is examined through the analysis of standard deviation's reduction.

C. Development of Components and System Descriptions

The system development phases for automated semantic-based evaluation are detailed in Table I. Each phase describes the student response trajectory, encompassing code submission, Abstract Syntax Tree (AST) formation, canonicalization, structural comparison, execution-integration validation, score construction, and expert agreement evaluation. Table I summarizes each development phase, its corresponding main component, and its description

TABEL I
SYSTEM DEVELOPMENT STAGES

Phase	Component	Description
1	FIB Input	Students fill in missing code segments using Python strings.
2	AST Builder	Student responses are parsed into ASTs using Python's ast module.
3	AST Canonicalisation	Semantic transformation is applied to eliminate syntactic noise and extract logical patterns.
4	AST Comparison	Student ASTs are compared to reference keys using tree isomorphism and subtree matching.
5	Output Execution	Code is executed in a sandbox environment; outputs are compared using hash functions.
6	Scoring	Final scores are computed: 70% from AST similarity, 30% from output equivalence.
7	Expert Validation	System scores are compared with manual expert annotations using Cohen's Kappa.
8	Accuracy Evaluation	Precision, recall, F1-score, and false negative rate are calculated against expert ground truth

To fulfil the educational goals of Fill-in-the-Blank (FIB) activities, a 70:30 ratio distribution between the Abstract Syntax Tree (AST) structure and the execution output score has been implemented. This approach emphasises the logical structure of students' code rather than concentrating exclusively on the output. The AST section determines whether students produce a semantically equivalent logic, and the execution output serves as a layer for functional verification. This scoring ratio may be modified in the case of open-ended projects to best appreciate functional realism [17].

The process of canonicalisation includes the removal of syntactic metadata (linenos, col_offsets, and ctx), normalisation of local variable names, lexicographic ordering of unordered code blocks, and the syntactic restructuring of control flows, such as the transformation of for loops to while loops, when their logical meanings are semantically equivalent. Consider the following example:

```
# Original
for i in range(n):
    total += i

# After transformation
i = 0
while i < n:
    total += i
    i += 1
```

This transformation reduces syntactic noise and extracts stable logical patterns which allow the AST to be compared man_

D. Validation and Analysis of FIB Responses

As part of the validation of the system, the analyses of 200 FIB responses were conducted where 2 experts performed independent analysis of the responses. These responses were classified into four types: Fully correct (about 35%), Syntactic (about 15%), Minor Semantic (about 25%), and Logical or Output (about 25%). Inter-rater reliability for 50 responses was calculated and yielded a value of Cohen's Kappa 0.88. Data analysis occurred on three levels. First, dimensions of reliability and accuracy were assessed using Cohen's Kappa with the other measures of precision, recall, F1-score, and the False Negative rate. Second, the alignment of the system-generated scores and the expert-assigned scores were evaluated using Pearson correlation. Third, qualitative error analysis was conducted focusing on the false negatives and false positives to expose weaknesses in the canonicalisation and subtree matching algorithms. These analyses provided the basis for system refinements.

E. Ethics and Reproducibility

This study was ethically approved by the programming faculty within the Department of Information Technology at the State Polytechnic of Malang. To protect confidentiality, all participant data were anonymised. The open-access GitHub repository contains the source code for the AST Evaluator, the anonymised dataset, and the scripts for the statistical analyses conducted. This is to facilitate independent verification and to provide transparency around the methods employed, in the spirit of open science [19], [20].

III. Result And Discussion

This part explains the outcomes derived from the three phases of the study: (1) technical validation along with system acceptance, (2) analysis of the pedagogical effectiveness, and (3) epistemological evaluation of the evaluator based on AST.

A. Technical Validation and System Acceptance

Students, instructors, and administrators engaged in a black-box testing procedure which ascertained that all features of the system function as expected. The system met the operational feasibility minimum requirements. Table II provides the results of black-box testing.

TABEL II
Black-Box Testing Results

User Group	Features Tested	Result
Students	Course, Module, FIB, Test, Results	Valid
Instructors	Content Management & Automated FIB Items	Valid
Administrators	Data Management & Reporting	Valid

The System Usability Scale (SUS) evaluates user satisfaction with the developed system [21], [22]. Table III presents the distribution of SUS scores including the range, number of respondents, and the respective scores. In assessing subjective usability and system acceptance, the SUS questionnaire was distributed. Table III summarizes the distribution of 35 respondent scores, resulting in an average score of 76.47, which strongly indicates the functional feasibility of the system.

TABEL III
Distribution of SUS Scores

SUS Score Range	Number of Respondents	Individual Scores
40–49	1	45
50–59	6	50, 52.5, 55, 55, 57.5, 57.5
60–69	3	65, 65, 67.5
70–79	7	70, 70, 72.5, 75, 75, 77.5, 77.5
80–89	10	80, 80, 82.5, 82.5, 82.5, 85, 85, 85, 87.5, 87.5
90–100	8	90, 90, 90, 92.5, 92.5, 95, 95, 100, 100, 100

In order to illustrate the system's effectiveness in resolving the false negative issues discussed in the introduction, a comparison was made with a string matching–based system that used the same dataset.

Previous analyses suggest substantial difficulty with the identification of semantic equivalence with string matching methods, leading to false negative rates of over 20% during static evaluations[23]. In contrast, the current system noticeably improved the false negative reduction and alignment with expert annotations, achieving substantial agreement as measured by Cohen's Kappa [24].

FIB and AST evaluator achieved a Cohen's Kappa of 0.83 compared to a value of 0.61 with string matching, with false negative rates of 21.5 and 4.5% respectively as shown in Table IV. The results suggest the integrated AST technology identifies semantic equivalence which string based evaluations fail to achieve, leading to greater accuracy and consistency in evaluations.

B. Pedagogical Effectiveness Analysis

B.1. Between-Group Comparison

An Independent Sample T-Test framework was employed for evaluations over the three measurement periods of Pre-Test, Post-Test, and Delay-Test. These results are summarised in Table IV.

TABLE IV
Summary of Statistical Analysis with Significance Indicators

Group	Test Phase	Mean Score	t-value	p-value
Experimental	Pre-Test	27.45	−0.046	0.963
Control	Pre-Test	27.99		
Experimental	Post-Test	92.92	4.488	0.000
Control	Post-Test	49.68		
Experimental	Delay-Test	81.84	3.387	0.001
Control	Delay-Test	49.52		

Experimental	Pre → Post	27.45 → 92.92	7.342	0.000
Experimental	Post → Delay	92.92 → 81.84	1.780	0.087
Control	Pre → Post	27.99 → 49.68	2.019	0.055
Control	Post → Delay	49.68 → 49.52	0.014	0.989

The Independent Sample T-Test demonstrated statistically significant improvement for the experimental group (0.000) and significant retention within the Delay-Test phase (0.001). The intervention effect size was very large (Cohen's d = 1.82), classified as "very large effect" according to Kraft's criteria. A Normalized Gain of 0.90 indicates a substantial increase in conceptual understanding [25]. There was a reduction in the standard deviation from 39.48 (Pre-Test) to 13.20 (Post-Test) suggesting increased homogenisation of student comprehension within the experimental group.

B.2. Multiple Regression and Gain Contextualisation

The multiple regression analysis targeted the predictive impact of each variable on the Post-Test score improvement. Among the AST structural score components, the strongest contribution was noted in the structural scoring AST component compared to execution output ($\beta = 0.28$) and response time ($\beta = 0.12$) with ($\beta = 0.61$, $p < 0.01$). Results of this analysis substantiate the importance of semantic fairness most prominently AST scoring in understanding student improvement.

The large gain ($g = 0.90$) is to be interpreted with caution as this reflects two conditions. One includes the control group's Post-Test lower baseline performance (49.68) suggesting issues with assessing initial performance. The second is that the FIB+AST system functioned as a highly accurate remedial feedback tool and effectively removed semantic penalties of false negatives. Literature describes such sizable gains as typical in intensive remedial programmes [26].

C. AST Evaluator Validation

C.1. Scoring Weight Trade-Off

For a scoring scheme of 70:30 (AST:Output) and two alternatives the analysis focused on the trade-off between false negatives (FNR) and false positives (FP). The 70:30 scheme yielded the lowest FNR, despite a slightly higher FP rate. A 70:30 ratio was adopted based on axiological principles of assessment: unfair penalties (false negatives) are more psychologically and pedagogically damaging than minor recognition errors (false positives) [27].

C.2. Limitations of Canonicalization and Tree Isomorphism

The AST Evaluator had an F1-Score of 0.966 and thus, high classification accuracy. Yet, the presence of false negatives revealed the system's ontological limitations: recursive structures, nested lambda functions, and nested ternary operators were not aligned with canonical structures. Such limitations stem from the rigidity of tree isomorphism, which strictly analyzes the framework of interdependencies, while Abstract Syntax Trees (ASTs) are primarily Directed Acyclic Graphs (DAGs) [28]. Consequently, tree isomorphism does not capture the abstract semantic relationships anchored on control and data flow. According to Dikici et al. [29], semantic code recognition based on AST edit distance is ineffective due to structurally different code.

D. Ontological and Epistemological Analysis

D1. Ontological Foundation: Moving from Syntax to Relational Representation

This study adopts the perspective of Ontic Structural Realism (OSR) with the understanding that program logic constitutes a key structural reality. At first the Abstract Syntax Tree (AST) was treated as an ontological set as a representation of the syntactic structure. However, in the evaluation, the reliance on the AST as an adequate representation was shown to be an overestimation. The canonicalization failures associated with recursive constructs indicate that the tree isomorphism thesis fails to adequately capture the essential structural reality of program logic.

To address these issues, this study proposes an ontological shift towards graph-based models, particularly Control Flow Graphs (CFG) and Data Flow Graphs (DFG). This shift recognises that relational ontologies consisting of control flow and data dependencies are a more accurate reflection of program logic than surface syntax. The CFGs and DFGs not only embody vital semantic relations but also aid in pinpointing semantic equivalence despite varied coding approaches. The structural integrity that graph models provide also enhances the objectivity of system assessments, offering a potential for valid classification that bypasses syntactic alterations.

D.2. Epistemology: Post-Positivism and Uncertainty

The evaluation system is grounded in a post-positivist epistemology, one that treats inference to the truth as non-absolute and probabilistic. The existencesss of a false positive (FP) finding in automated semantic validation demonstrates the intrinsic limitations of observation. Multiple outputs that are similar may not logically mean the same thing. This speaks to the uncertainty inherent in automated semantic validation. The empirical data was validated systemically through cross-triangulation :

a. Inferential Validity: Evaluated through internal consistency measures (F1-Score, Recall) of the AST and the system's accuracy in replicating correct structural decisions.

b. Pedagogical Validity: Evaluated through external reliability (expert annotations, κ), practical impact (Normalised Gain, g), and the return on investment measures (ROI).

TABLE V
VALIDATION OF PEDAGOGICAL FEASIBILITY

Analytical Aspect	Key Findings	Quantitative Validation	Philosophical Validation
Technical Validity & Usability	FIB+AST system is functional and exhibits high usability.	SUS = 76.47; FNR = 4.5%	AST serves as a partial structural representation prior to expansion.
Pedagogical Effectiveness	Significant learning gains supported by fairness validation.	Cohen's d = 1.82; g = 0.90; β_AST = 0.61	Semantic accuracy ensures evaluative fairness; promotes transfer learning.
AST Evaluator Performance	High inferential accuracy and strong expert reliability.	F1-Score = 0.966; κ = 0.83	Inference is probabilistic; demands epistemic transparency via confidence score.

A Confidence Score, which is based on Tree Edit Distance, is produced and used to calibrate inferences. This score increases transparency and aids decision-making, offering a form of control to the user in cases of extreme uncertainty and manual validation. Thus, the system is a calibrated, inferential instrument of revision and not an absolute one.

The following Table V captures key findings, based on the post-positivist epistemological paradigm, which demonstrates the technical feasibility and pedagogical impact of the AST-Graph based evaluation system while outlining its pedagogical impact within the probabilistic framework of the system.

IV. CONCLUSION

A. Summary

This study provides two main contributions. First, technically, the AST-based model significantly lowers the false negative rate from 21.5% (observed in string matching) to 4.5%. This indicates that AST can be considered an effective semantic filter for the detection of logical equivalence in student answers. Second, pedagogically, semantic accuracy stands out as the foremost factor in driving improvements in significant learning. The size of the impact (Cohen's d = 1.82) and the multiple regression analysis determine that pure AST scores most strongly predict increased comprehension.

The AST-based semantic evaluation model demonstrates high construct validity and construct reliability. It surpasses the traditional syntactic methods with its validated results and excellent classification results (F1 = 0.966; κ = 0.83). A validated triangulation framework provides a solid epistemic base. The inferences a system generates can be treated as probabilistic ground truth, permitting teachers to spend less time manual grading and more time on the analysis of ambiguous cases.

AST is, ontologically, a valid semantic representation. However, the limitations of tree isomorphism in recognizing recursive structures and nested lambda functions reveal the need to attentively improve nested tree structures for AST to achieve its full potential as a more powerful semantic analysis model. In its axiological dimension, the system reveals functional feasibility and solid user acceptance, illustrated by a System Usability Scale (SUS) score of 76.47, which refers to extended usability. Usability and user trust are furthered by features such as AST visualization and confidence scoring.

Moreover, the AST model has a quantifiable pedagogical effect. A normalized gain of 0.90 with a low standard deviation suggests some degree of conceptual homogenization. Consistent retention outcomes suggest that semantic fairness—not feedback speed—primarily aids in the transfer of learning.

B. Recommendations

Future research should expand the scope of semantic evaluation. It would be beneficial to use graph isomorphism techniques based on Control Flow Graphs (CFG) and Data Flow Graphs (DFG) to improve upon the limitations of tree isomorphism. More complex problems could include detecting semantic errors, performing dynamic code analysis, and structural assessments across various programming languages. Generalization of the AST would also need to be assessed on other languages, such as Java or C++. Future research could also explore the use of Large Language Models (LLMs) to enable flexible and situational semantic evaluations. Longitudinal studies are recommended to assess long-term retention of learning outcomes. Implementing additional Delay-Tests at intervals of one and three months would help measure the durability of learning effects. Additionally, it would be valuable to correlate student activity logs with score homogenization to identify effective interaction patterns. In line with open science principles, it is strongly encouraged to publish anonymized datasets and the source code for the AST Evaluator in open repositories. This practice promotes external validation and ensures the reproducibility of results.

REFERENCES

[1] A. Agarwal, "Python for CS1, CS2 and beyond," *J. Comput. Small Coll.*, vol. 20, pp. 262–270, Jan. 2005.

[2] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated Grading and Feedback Tools for Programming Education: A Systematic Review," *ACM Trans. Comput. Educ.*, vol. 24, no. 1,

Feb. 2024, doi: 10.1145/3636515.
[3] Z. Fan, S. H. Tan, and A. Roychoudhury, "Concept-Based Automated Grading of CS-1 Programming Assignments," *ISSTA 2023 - Proc. 32nd ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 199–210, 2023, doi: 10.1145/3597926.3598049.
[4] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Comput. Educ.*, vol. 41, pp. 121–131, Sep. 2003, doi: 10.1016/S0360-1315(03)00030-7.
[5] M. Erfan, I. Handika, Afriyanti, W. Aziiz Hari Mukti, and T. Ratu, "Penggunaan Bahasa Pemrograman Python dalam Analisis Hubungan Peminat dan Daya Tampung Seluruh Prodi di Indonesia Pada PTN Akademik, Vokasi dan PTKIN Tahun 2023," *J. Classr. Action Res.*, vol. 6, no. 2, pp. 313–9, 2024, [Online]. Available: http://jppipa.unram.ac.id/index.php/jcar/index
[6] A. Kholik, H. Bisri, Z. K. Lathifah, B. Kartakusumah, M. Maufur, and T. Prasetyo, "Impelementasi Kurikulum Merdeka Belajar Kampus Merdeka (MBKM) Berdasarkan Persepsi Dosen dan Mahasiswa," *J. Basicedu*, vol. 6, no. 1, pp. 738–748, 2022, doi: 10.31004/basicedu.v6i1.2045.
[7] Z. Swilam, A. Hamdy, and A. Pester, "Improving code semantics learning using enhanced Abstract Syntax Tree," *Int. J. Comput. Appl.*, vol. 47, no. 1, pp. 57–69, Jan. 2025, doi: 10.1080/1206212X.2024.2443506.
[8] Z. Zhu, N. Funabiki, M. Mentari, S. T. Aung, W. C. Kao, and Y. F. Lee, "An Automatic Code Generation Tool Using Generative Artificial Intelligence for Element Fill-in-the-Blank Problems in a Java Programming Learning Assistant System," *Electron.*, vol. 14, no. 11, pp. 1–27, 2025, doi: 10.3390/electronics14112261.
[9] A.-T. P. Nguyen and V.-D. Hoang, "Development of Code Evaluation System based on Abstract Syntax Tree," *J. Tech. Educ. Sci.*, vol. 19, no. 1, pp. 15–24, 2024, doi: 10.54644/jte.2024.1514.
[10] D. R. Fudholi and A. Capiluppi, "Artificial intelligence for source code understanding tasks: A systematic mapping study," *Inf. Softw. Technol.*, vol. 189, p. 107915, 2026, doi: https://doi.org/10.1016/j.infsof.2025.107915.
[11] Geetika, N. Kaur, and A. Kaur, "A Semantic-driven approach to detect Type-4 code clones by using AST and PDG," *Int. J. Inf. Technol.*, Jul. 2025, doi: 10.1007/s41870-025-02670-2.
[12] M. Hammad, Ö. Babur, H. Basit, and M. Brand, "Clone-Seeker: Effective Code Clone Search Using Annotations," *IEEE Access*, vol. 10, p. 1, Jan. 2022, doi: 10.1109/ACCESS.2022.3145686.
[13] P. R., T. Mg, and J. Kannimoola, "Automated Code Assessment and Feedback: A Comprehensive Model for Improved Programming Education," *IEEE Access*, vol. PP, p. 1, Jan. 2025, doi: 10.1109/ACCESS.2025.3554838.
[14] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, "Automatic Grading and Feedback using Program Repair for Introductory Programming Courses," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, in ITiCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 92–97. doi: 10.1145/3059009.3059026.
[15] G. Jiang, "Design and Implementation of an Automatic Grading System for Programming Code Based on Artificial Intelligence," in *2025 IEEE 3rd International Conference on Image Processing and Computer Applications (ICIPCA)*, 2025, pp. 1846–1851. doi: 10.1109/ICIPCA65645.2025.11139057.
[16] E. Telli and A. Altun, "Effect of semantic encoding strategy instruction on transfer of learning in e-learning environments," *J. Educ. Technol. Online Learn.*, vol. 6, Jan. 2023, doi: 10.31681/jetol.1205276.
[17] A. Sheoran *et al.*, "Data reporting quality and semantic interoperability increase with community-based data elements (CoDEs). Analysis of the open data commons for spinal cord injury (ODC-SCI)," *Exp. Neurol.*, vol. 385, p. 115100, 2025, doi: https://doi.org/10.1016/j.expneurol.2024.115100.
[18] C. Xu, M. B. Mashhadi, Y. Ma, R. Tafazolli, and J. Wang, "Generative Semantic Communications With Foundation Models: Perception-Error Analysis and Semantic-Aware Power Allocation," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 7, pp. 2493–2505, 2025, doi: 10.1109/JSAC.2025.3559120.
[19] Jessica J Santana and Seonghoon Kim, "From Values to Codes: A computational text analysis of the codification of occupational ethics," *Organ. Stud.*, p. 01708406251317255, Feb. 2025, doi: 10.1177/01708406251317255.
[20] A. Brockinton, M. Salnitri, F. Kooner-Evans, J. McAlaney, and S. Thompson, "An exploratory study on the human component using a cultural model to define open research topics for secure socio-technical systems," *Technol. Soc.*, vol. 83, p. 103000, 2025, doi: https://doi.org/10.1016/j.techsoc.2025.103000.
[21] Macclarck Pessoa Nery, Severiano José dos Santos Neto, Roberty Santos Alves, João Vitor dos Santos Santana, Sandro Griza, and Carlos Otávio Damas Martins, "Development of educational software for stainless steel selection and evaluating usability using the System Usability Scale (SUS)," *Int. J. Mech. Eng. Educ.*, vol. 53, no. 4, pp. 957–972, Aug. 2024, doi: 10.1177/03064190241266978.
[22] S. F. Brähmer *et al.*, "Development of a Serious Game App (Digimenz) for Patients With Dementia: Prospective Pilot Study for Usability Testing in Inpatient Treatment and Long-Term Care," *JMIR Serious Games*, vol. 13, p. e69812, 2025, doi: 10.2196/69812.
[23] X. Xu *et al.*, *MGF-ESE: An Enhanced Semantic Extractor with Multi-Granularity Feature Fusion for Code Summarization*, vol. 1, no. 1. Association for Computing Machinery, 2025. doi: 10.1145/3696410.3714544.
[24] L. Deng, X. Ren, C. Ni, M. Liang, D. Lo, and Z. Liu, "Enhancing Project-Specific Code Completion by Inferring Internal API Information," *IEEE Trans. Softw. Eng.*, vol. 51, no. 9, pp. 2566–2582, 2025, doi: 10.1109/TSE.2025.3592823.
[25] D. Chicco, A. Sichenze, and G. Jurman, *A simple guide to the use of Student's t-test, Mann-Whitney U test, Chi-squared test, and Kruskal-Wallis test in biostatistics*, vol. 18, no. 1. BioMed Central, 2025. doi: 10.1186/s13040-025-00465-6.
[26] Chengliang Wang, Xiaojiao Chen, Yifei Li, Pengju Wang, Haoming Wang, and Yuanyuan Li, "MetaClassroom: A New Paradigm and Experience for Programming Education," *J. Educ. Comput. Res.*, vol. 63, no. 4, pp. 864–901, Feb. 2025, doi: 10.1177/07356331251322470.
[27] H. Cui, M. Xie, T. Su, C. Zhang, and S. H. Tan, "An Empirical Study of False Negatives and Positives of Static Code Analyzers From the Perspective of Historical Issues," vol. 1, no. 1, pp. 1–26, 2024, [Online]. Available: http://arxiv.org/abs/2408.13855
[28] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with GNNs," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019.
[29] S. Dikici and T. T. Bilgin, "Advancements in automated program repair: a comprehensive review," *Knowl. Inf. Syst.*, vol. 67, no. 6, pp. 4737–4783, 2025, doi: 10.1007/s10115-025-02383-9.