International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

CodeBuddy: A Browser-Based Platform for Learning Data Structures and Algorithms Using AI-Driven Feedback

K. Satya Krishna Chowdary[1], P. Sowjanya[2], M. Anish Kumar[3],N.Ramanaiah[4]
[1234]Bachelor of Technology, Department of Computer Science and Engineering,
Bapatla Engineering College, Bapatla – 522101, Andhra Pradesh, India
Under the Guidance of Dr. P. S. V. Vachaspati,Professor, Department of CSE, Bapatla Engineering College

Abstract*—Data Structures and Algorithms (DSA) remains one of the most challenging subjects in undergraduate computer science education. Students frequently struggle not because the concepts are inherently impossible, but because the feedback available to them is either absent or counterproductive. Most online platforms report only whether code passed or failed without explaining why, while unrestricted AI tools tend to hand over complete answers before students have had a genuine opportunity to think. This paper presents CodeBuddy, a lightweight, browser-based DSA practice environment that uses the DeepSeek-Coder large language model to provide intelligent, structured assistance to learners as they work through problems. The platform hosts 70 carefully designed problems spanning 7 core topics, with support for Python, JavaScript, Java, C, and C++. Its central feature is an attempt-gating mechanism: students must submit at least two attempts before hints become available, and at least four before a full solution is accessible. This design draws on established learning science research and is intended to encourage genuine engagement with each problem rather than reaching for help at the first sign of difficulty. The system additionally provides real-time error analysis, time and space complexity feedback, and a six-tier progressive hint system that guides thinking without replacing it. Early observations indicate that the graduated approach nudges students toward deeper reasoning before assistance is sought.*

Keywords—Data Structures and Algorithms, AI-Powered Education, Large Language Models, Progressive Hints, Code Validation, DSA Practice Platform, Productive Failure, DeepSeek-Coder

I. INTRODUCTION

Data Structures and Algorithms occupies a central position in undergraduate computer science education, and this status is well deserved. Major technology companies structure a significant portion of their technical hiring processes around DSA problem solving, and the capacity to reason algorithmically, evaluate time and space trade-offs, and select appropriate data structures for a given problem distinguishes a genuinely capable software engineer from one who has merely learned syntax. Despite this importance, a substantial proportion of students find DSA difficult to study on their own, and the tools available to support them have meaningful gaps.

Three recurring problems make independent DSA practice frustrating. Students often have no reliable way to know whether their solution is actually correct before submitting it formally. When they get stuck, no clear path forward exists. And when they seek external help, they are usually met with one of two unhelpful extremes: either nothing useful, or the complete answer. Platforms such as LeetCode and HackerRank provide verdict-based feedback—pass or fail—which is adequate for competitive practice but poorly suited to the early stages of developing understanding. A wrong answer verdict tells a student very little: not which part of the logic failed, not whether the problem is a typo or a fundamental error, and not where to begin looking.

Advances in large language models have opened a genuine opportunity to address these shortcomings. Code-trained models such as DeepSeek-Coder can read student submissions, trace through execution, identify specific

errors, offer targeted hints, and produce full working solutions with explanations. The risk, documented across educational research, is that most tools built on these models simply answer the question directly. The model becomes a shortcut, and the productive struggle that leads to durable understanding never occurs.

CodeBuddy is built around the principle that assistance should be earned through effort rather than freely available on request. It is a single-page web application built with HTML, CSS, and JavaScript, connected to the DeepSeek-Coder API for its AI capabilities. The platform enforces a graduated help system: error analysis is available after any submission, hints unlock after two failed attempts, and full solutions become accessible only after four. Every feature traces back to a body of learning science research showing that students who wrestle with a problem before receiving help develop stronger, more lasting understanding.

II. RELATED WORK

A. Online Judge Platforms

LeetCode [1], HackerRank [2], and Codeforces [3] are among the most widely used platforms for coding practice. Their shared model of submitting code against hidden test cases and receiving a pass or fail verdict works well for competitive preparation but leaves students with limited guidance when they are genuinely learning. A wrong answer result tells a student almost nothing useful—not which test case failed, not the nature of the error, and not what direction to take next. The platforms fulfil their stated purpose at the point of returning a verdict; helping students build understanding is outside their design.

B. Intelligent Tutoring Systems

Intelligent Tutoring Systems have been studied as a vehicle for personalised learning support for several decades [4]. Systems such as AutoTutor and ALEKS can adapt to individual learners and provide structured, step-by-step feedback. A well-built ITS for DSA could theoretically offer precisely the scaffolding students need. In practice, constructing such a system demands substantial domain modelling effort and tends to produce tools narrowly confined to a single language or topic area. Wide adoption in algorithm education has not materialised as a result.

C. Large Language Models in Education

Large language models have demonstrated strong performance on coding tasks [5]. Tools such as GitHub Copilot and ChatGPT can write complete programs, trace logic, explain errors, and respond to almost any programming question. The educational concern with giving students unrestricted access to these tools is well established: when a model can instantly produce a correct solution, students have little incentive to struggle, and that struggle is precisely what produces durable learning. Research on productive failure [6] makes this point consistently and clearly.

D. Gap in Existing Tools

No existing tool combines a curated DSA problem set, multi-language support, AI-driven code validation, a progressive hint system tied to attempt count, error analysis, and complexity feedback in a single application that runs directly in the browser without installation. CodeBuddy is designed to address that specific combination of needs.

III. LITERATURE SURVEY

This section surveys the prior work that most directly shaped CodeBuddy's design. The five areas covered are: online coding practice platforms, intelligent tutoring systems in computer science education, large language models in programming contexts, code validation approaches, and educational psychology research on learning through structured difficulty.

A. Online Coding Practice Platforms

LeetCode [1], HackerRank [2], and Codeforces [3] define the current standard for online DSA practice. Keuning et al. [8] conducted a systematic review of automated feedback tools for programming exercises and found that the large majority provide only correctness verdicts with almost no tools offering explanations or targeted guidance. Price et al. [9] demonstrated that immediate, specific feedback produces meaningfully better learning outcomes in introductory CS courses compared to delayed or generic feedback. These two findings together describe both the gap in current platforms and the opportunity CodeBuddy addresses.

B. Intelligent Tutoring Systems in Computer Science Education

VanLehn [4] conducted a landmark meta-analysis showing that well-designed intelligent tutoring systems can produce learning gains approaching those of one-on-one human tutoring, providing strong motivation for building AI-powered educational tools. Research on systems such as KOLI [10] demonstrates that step-by-step feedback in programming contexts improves student retention. Crow et al. [11] found that adaptive feedback significantly reduces dropout rates in data structures courses. Aleven et al. [12] identified the central limitation: building a real ITS requires extensive domain authoring work and typically covers only a narrow topic. CodeBuddy bypasses this entirely by using a pre-trained LLM whose reasoning generalises across problem types and programming languages without additional modelling effort.

C. Large Language Models in Programming Education

Chen et al. [5] introduced Codex and demonstrated its ability to solve a meaningful portion of introductory programming problems. Austin et al. [13] extended this to a wider range of programming tasks. Sarsa et al. [14] used GPT-3 to generate programming exercises and found that prompt design was critical to preventing the model from over-assisting students—a finding that directly influenced CodeBuddy's system prompt strategy. Kazemitabar et al. [15] observed that ChatGPT, while technically accurate, tends to solve problems outright rather than guiding students through them. DeepSeek [7] demonstrated that code-specialised models outperform general-purpose ones on code reasoning tasks. CodeBuddy uses DeepSeek-Coder with carefully engineered system prompts designed to produce diagnostic feedback and incremental hints, deferring full solutions until after the attempt threshold is reached.

D. Code Validation and Static Analysis

Traditional educational code validation relies on test-case execution. Johnson [16] reviewed static analysis tools for educational use and found them effective for detecting syntax and simple logic errors but difficult to configure across varied assignment types. Hsiao et al. [17] showed that combining static analysis with dynamic execution produces more actionable feedback than either method in isolation. Rivers and Koedinger [18] built a data-driven hint system drawing on historical solution paths from previous students—a strong approach that requires a large solution archive before it can function. CodeBuddy uses AI-based semantic validation, which requires no historical data and generalises immediately to new problems across all supported languages.

E. Educational Psychology: Productive Failure and Desirable Difficulty

The attempt-gating mechanism in CodeBuddy draws its justification from a surprisingly coherent body of educational psychology literature. Kapur [6] introduced productive failure through a series of controlled classroom experiments, showing that students who grapple with problems independently—before any instruction is given—tend to build more flexible and lasting understanding than peers who receive guidance upfront. Schwartz and Bransford [19] offered a complementary finding: exploratory effort before formal teaching activates prior knowledge in ways that make the subsequent instruction far more meaningful. Koedinger and Aleven [20] named the underlying tension the assistance dilemma, noting that too much help undermines learning while too little drives students away—CodeBuddy's two- and four-attempt thresholds are a direct attempt to find a workable middle ground. Bjork [21] articulated the broader principle through the concept of desirable difficulties: friction in the learning process is not an obstacle but a mechanism through which durable memory is built. Loibl et al. [22]

meta-analysed the productive failure literature and found the effect most reliable when the initial struggle is followed by well-structured feedback, which is exactly the role the platform's AI analysis plays. Beyond individual struggle, Vygotsky's concept of the Zone of Proximal Development [24] highlights that learners progress most effectively when scaffolded just beyond their current capability—a principle that directly supports staging hints incrementally rather than offering them all at once. Denny et al. [25] examined how students interact with AI code generation tools in real coursework and found that many rely on them as answer engines rather than learning aids, a pattern that reinforces the need for gating mechanisms like those in CodeBuddy. Loibl et al. [22] conducted a meta-analysis confirming the productive failure effect and noting that its benefits are most pronounced when structured feedback follows the period of struggle—precisely the role CodeBuddy's AI layer is designed to fulfil. Prathyusha et al. [23] demonstrated a related application of real-time AI feedback in a traffic sign recognition system combining YOLOv8 with multilingual voice alerts, reinforcing how adaptive, immediate feedback mechanisms translate across different educational and applied domains.

Table III: Literature Survey — Key Works and Relation to CodeBuddy

Ref	Author/Year	Key Idea	Relation to CodeBuddy	CodeBuddy Advantage
[1]	LeetCode (2024)	Pass/fail online judge	Adds AI diagnosis on top of correctness	Multi-level: errors, hints, complexity beyond verdict
[2]	HackerRank (2024)	Competitive coding platform	Shows scope of DSA platforms	Curriculum-based learning over competitive ranking
[3]	Codeforces (2024)	Contest-focused judge	Online judge baseline	Focuses on understanding over competition
[4]	VanLehn (2011)	ITS rivals human one-on-one tutoring	Motivates AI-based DSA tutoring	ITS-level feedback without hand-crafted domain models
[5]	Chen et al. (2021)	Codex solves programming tasks broadly	Confirms LLMs understand code	LLM used for guidance, not free answer delivery
[6]	Kapur (2016)	Struggling before instruction deepens learning	Core rationale for attempt-gate	2-attempt and 4-attempt locks enforce productive struggle
[7]	DeepSeek (2024)	Code-specialised LLM outperforms general ones	Justifies DeepSeek-Coder choice	Lightweight, cost-effective API, no server needed
[8]	Keuning et al. (2018)	Most tools give correctness verdicts only	Identifies the feedback gap addressed	Provides errors, hints, complexity, full solutions
[9]	Price et al. (2017)	Targeted feedback improves CS1 outcomes	Supports real-time validation design	Instant specific feedback in 5 languages
[10]	Brusilovsky (2003)	Step-by-step hints improve retention	Validates progressive hint approach	Hints generated dynamically; no pre-authored trees

[11]	Crow et al. (2018)	Adaptive feedback reduces DS dropout	Relevant to CodeBuddy's DSA focus	All 7 topics with progressive difficulty filtering
[12]	Aleven et al. (2016)	ITS requires heavy domain authoring	Highlights LLM advantage over ITS	Zero domain authoring; AI generalises automatically
[13]	Austin et al. (2021)	LLMs generate correct code broadly	Confirms AI backend reliability	Solution shown only after 4 failed attempts
[14]	Sarsa et al. (2022)	Prompt design critical for LLM exercises	Validates careful prompt engineering	Strict prompts ensure hints before threshold
[15]	Kazemitabar et al. (2023)	ChatGPT solves directly without guiding	Shows risk of uncontrolled LLM access	Attempt-gate prevents early solution access
[16]	Johnson (2002)	Static analysis needs configuration	Contextualises AI-based validation	No config; AI adapts to any problem and language
[17]	Hsiao et al. (2017)	Static + dynamic gives better feedback	Supports hybrid validation approach	Validation, error analysis, and complexity in one flow
[18]	Rivers & Koedinger (2017)	Data-driven hints from history are effective	Motivates hint-based learning	No historical dataset; works for new problems instantly
[19]	Schwartz & Bransford (1998)	Prior exploration activates learning structures	Basis for requiring attempts first	Progressive unlocking mirrors explore-then-learn cycle
[20]	Koedinger & Aleven (2007)	Assistance dilemma: balance help carefully	Motivates 2-attempt and 4-attempt gates	Help level calibrated to minimise over-assistance
[21]	Bjork (1994)	Desirable difficulties improve long-term retention	Justifies deliberate friction in help access	Attempt gates create difficulty without frustration
[22]	Loibl et al. (2017)	Productive failure + structured feedback confirmed	Validates full attempt + feedback design	Combines productive failure gate with AI feedback
[23]	Prathyusha et al. (2026)	Real-time traffic sign detection with YOLOv8 and multilingual voice alerts	Demonstrates real-world AI + voice feedback integration	Confirms AI feedback architecture applicability in educational systems
[24]	Vygotsky (1978)	Zone of Proximal Development: learning progresses fastest when scaffolded just	Justifies staged hint release — each tier pushes learner just beyond current understanding	Six-tier hint system mirrors ZPD scaffolding; no hint reveals more than one conceptual step at a time

		beyond current ability		
[25]	Denny et al. (2023)	Students use GPT-3 as answer engine rather than learning tool without structural guardrails	Supports attempt-gate rationale: uncontrolled LLM access leads to passive answer consumption	Attempt locks prevent zero-effort solution access; AI is guide not answer dispenser

IV. SYSTEM DESIGN AND ARCHITECTURE

A. Overview

CodeBuddy is a fully client-side web application that requires no installation and no back-end server beyond the DeepSeek-Coder API. The decision to package the entire application as a single HTML file containing all CSS, structure, and JavaScript logic was deliberate: a zero-install tool that opens immediately in any browser is far more likely to be used consistently than one with setup requirements. The interface is divided into three main regions: a left sidebar for navigation and problem selection, a centre panel containing the code editor, and a right panel for AI-generated output.

B. Problem Dataset

The platform includes 70 DSA problems distributed across 7 core topic areas. Each problem provides a clearly worded statement, sample input and expected output, and pre-filled function signatures for all five supported languages. The distribution of difficulty levels within each topic was designed to give students a reasonable progression, with more challenging problems concentrated in areas such as Graphs and Heaps where conceptual depth is greater.

Table I: Problem Distribution by Topic and Difficulty

Topic	Easy	Medium	Hard	Total
Arrays	5	4	1	10
Strings	5	3	2	10
Linked List	3	5	2	10
HashMap	5	4	1	10
Graphs	1	6	3	10
Trees	4	4	2	10
Heaps	0	6	4	10
Total	**23**	**32**	**15**	**70**

C. Multi-Language Support

Every problem in the platform supports Python, JavaScript, Java, C, and C++. When a student selects a language, the editor is pre-populated with the correct function signature and the file header updates to reflect the selected language. The platform also performs basic language detection: if the code a student writes does not match the selected language based on keyword analysis, the discrepancy is flagged.

D. The Code Editor

The editor is built on a standard HTML textarea element and includes synchronised line numbers that scroll alongside editor content, tab-key support that inserts four spaces for indentation, and pre-filled function signature templates. Students can begin writing logic immediately without setting up boilerplate code.

E. Attempt Tracking and Progressive Help System

This feature most directly reflects the platform's pedagogical philosophy. The system tracks how many times each student has submitted code for a given problem. Error analysis is available from the very first submission. The hints panel is locked until at least two genuine attempts have been made, and the full solution is locked until four attempts have been completed. Once a student reaches a correct answer, the hint and error analysis buttons are disabled, since diagnostic feedback at that point serves no meaningful learning purpose. The attempt count is displayed as a row of coloured indicators in the sidebar.

V. AI-POWERED FEATURES

A. AI Backend: DeepSeek-Coder

CodeBuddy uses the DeepSeek-Coder model through the DeepSeek API. The choice of a code-specialised model over a general-purpose one was based on performance: models trained heavily on code perform substantially better at tasks such as tracing execution, identifying logic errors, and explaining algorithmic approaches in terms a student can act on. Students supply their own API key, which is held in memory for the session and not stored anywhere permanently.

B. Code Validation

When a student submits their code, the platform sends the submission to the model along with the problem statement, sample input, expected output, and a strict instruction set. The model traces the code step by step and responds with a single verdict: correct or incorrect. Constraining the response to a single word prevents the model from entering explanatory mode prematurely and keeps the validation step unambiguous.

C. Complexity Analysis

When a student reaches a correct solution, the platform automatically requests a complexity analysis. The model returns a structured JSON object specifying time complexity, a one-line justification, space complexity, and a corresponding justification. These are rendered as two Big-O notation cards in the output panel, giving students immediate insight into the efficiency of their approach.

D. Progressive Hints

After two failed attempts, the hints panel becomes available. The model generates six hints ordered from the most abstract and general to the most specific. None of the hints include solution code. They are displayed as a 2x3 grid of cards, designed so that a student who works through the first three hints has a meaningful nudge toward the solution without having the answer handed to them.

E. Error Analysis

Error analysis is available from the first submission onward. When requested, the model examines the student's code and produces a structured diagnostic report covering syntax errors with line references, logical errors with plain-language explanations, and an overall summary comparing what the code currently does against what it should do. This report is one of the most practically valuable features in the platform, particularly for students who have not yet developed the habit of reading their own code critically.

F. AI Solution with Explanation

After four failed attempts, the full solution button becomes accessible. The model returns working code in the student's selected language along with a four-to-six sentence plain-English explanation of the approach taken and why it works. The explanation is written in prose rather than bullet points, which produces a more coherent account of the reasoning and tends to be easier to follow.

Table II: Summary of AI Features and Availability

Feature	Availability	AI Function
Code Validation	After every submission	Traces execution and assesses correctness
Complexity Analysis	On reaching correct answer	Returns time and space Big-O in JSON format
Progressive Hints (6)	After 2 failed attempts	Produces hints from broad to specific
Error Analysis	After any submission	Identifies syntax and logic-level mistakes
Full Solution + Explanation	After 4 failed attempts	Generates working code with plain-English explanation

VI. DISCUSSION

A. Comparison with Existing Tools

Compared to LeetCode or HackerRank, CodeBuddy offers substantially richer feedback at each stage. Those platforms report whether code passed; CodeBuddy identifies what is wrong and provides structured guidance toward improvement, without doing the improvement for the student. Compared to unrestricted access to ChatGPT or GitHub Copilot, CodeBuddy is considerably more controlled. A student cannot request the full answer after a single submission. The attempt gates ensure that every student who reaches the solution has already made four attempts, worked through error analysis, and engaged with a staged hint system.

B. Design Decisions and Trade-offs

Building CodeBuddy as a single HTML file with no back-end server was primarily a decision about accessibility. A tool that opens immediately in any browser is more likely to be adopted and used consistently than one requiring environment setup. The trade-off is that state—attempt counts and the API key—is lost when the page is refreshed, a real limitation that could be addressed with browser local storage in a future version. The choice of DeepSeek-Coder over GPT-4 was driven by both performance and cost: code-specialised models perform better on programming tasks, and DeepSeek's API pricing is substantially lower per call, which matters when students are covering their own usage costs.

C. Limitations

Several limitations in the current version deserve honest acknowledgment. Code is validated against only a single sample test case per problem, whereas a robust judge would run many hidden test cases including boundary conditions. LLM-based validation is not infallible; models can make tracing errors on complex code. There are no user accounts, so progress does not persist between sessions. The problem set is fixed at 70 problems and cannot be extended without directly editing the source file.

D. Future Work

Several improvements are planned. Adding multiple hidden test cases per problem would reduce false correct verdicts. Persisting progress via browser local storage, and eventually a lightweight server, would make the platform viable over a full course term. A difficulty progression system that recommends the next problem based on past performance would give the learning path more direction. Expanding the problem set to include Dynamic

Programming, Sorting Algorithms, and Backtracking would provide full DSA coverage. A timed contest mode would allow students to practise under conditions closer to a real technical interview. Sandboxed code execution in an actual runtime environment would substantially improve validation accuracy.

VII. CONCLUSION

This paper has presented CodeBuddy, a browser-based DSA learning platform powered by the DeepSeek-Coder large language model. The platform brings together 70 problems across 7 topics, supports 5 programming languages, and provides AI-driven code validation, error analysis, progressive hints, complexity feedback, and full solution explanations—all without requiring installation or a back-end server.

The central design principle—that assistance should be earned through effort rather than freely provided—shapes every feature of the platform. Locking hints behind two failed attempts and solutions behind four creates conditions in which students engage genuinely with each problem before receiving substantive help. This reflects a substantial body of research in educational psychology. Kapur's productive failure framework [6], Bjork's desirable difficulties theory [21], and the meta-analytic findings of Loibl et al. [22] consistently demonstrate that struggling with a problem before receiving structured instruction produces deeper understanding and stronger long-term retention than receiving instruction immediately.

CodeBuddy addresses a genuine gap in the landscape of tools available for DSA education. It combines the breadth of a curated problem set, the convenience of a zero-install application, and the reasoning capability of a modern AI model in a single accessible platform. Whether students are preparing for technical interviews or working through DSA concepts for the first time, CodeBuddy is designed to make the learning process more rigorous, more structured, and more productive.

REFERENCES

[1] LeetCode. (2024). LeetCode — The World's Leading Online Programming Learning Platform. https://leetcode.com

[2] HackerRank. (2024). HackerRank — Practice Coding and Prepare for Technical Interviews. https://www.hackerrank.com

[3] Codeforces. (2024). Codeforces — Competitive Programming Platform. https://codeforces.com

[4] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," Educational Psychologist, vol. 46, no. 4, pp. 197–221, 2011.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.

[6] M. Kapur, "Examining productive failure, productive success, unproductive failure, and unproductive success in learning," Educational Psychologist, vol. 51, no. 2, pp. 289–299, 2016.

[7] DeepSeek AI. (2024). DeepSeek-Coder: When the Large Language Model Meets Programming. https://github.com/deepseek-ai/DeepSeek-Coder

[8] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," ACM Transactions on Computing Education, vol. 19, no. 1, pp. 1–43, 2018.

[9] T. W. Price, R. Zhi, and T. Barnes, "Evaluation of a data-driven feedback algorithm for open-ended programming," in Proc. 10th Int. Conf. Educational Data Mining (EDM), pp. 192–197, 2017.

[10] P. Brusilovsky, "Developing adaptive educational hypermedia systems: From implementation technique to pervasive technology," Int. Journal of Artificial Intelligence in Education, vol. 13, no. 2–4, pp. 159–172, 2003.

[11] T. Crow, A. Luxton-Reilly, and B. Wuensche, "Intelligent tutoring systems for programming education: A systematic review," in Proc. 20th Australasian Computing Education Conference (ACE), pp. 53–62, 2018.

[12] V. Aleven, B. McLaren, J. Sewall et al., "Example-tracing tutors: Intelligent tutor development for non-programmers," Int. Journal of Artificial Intelligence in Education, vol. 26, no. 1, pp. 224–269, 2016.

[13] J. Austin, A. Odena, M. Nye, M. Bosma et al., "Program synthesis with large language models," arXiv preprint arXiv:2108.07732, 2021.

[14] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in Proc. ACM ICER, pp. 27–43, 2022.

[15] M. Kazemitabar, S. Lajoie, and T. Doleck, "Analysis of ChatGPT's potential as an intelligent tutoring system for programming," Computers and Education: Artificial Intelligence, vol. 4, p. 100163, 2023.

[16] B. Johnson, "Generating pedagogic programs and their explanations," Artificial Intelligence and Education, vol. 12, pp. 131–154, 2002.

[17] I. Hsiao, J. Huang, and F. Brusilovsky, "Dynamic exercise generation and student modeling for web-based C programming," in Proc. 18th ACM ITiCSE, pp. 183–188, 2017.

[18] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: A self-improving Python programming tutor," Int. Journal of Artificial Intelligence in Education, vol. 27, no. 1, pp. 37–64, 2017.

[19] D. L. Schwartz and J. D. Bransford, "A time for telling," Cognition and Instruction, vol. 16, no. 4, pp. 475–522, 1998.

[20] K. R. Koedinger and V. Aleven, "Exploring the assistance dilemma in experiments with cognitive tutors," Educational Psychology Review, vol. 19, no. 3, pp. 239–264, 2007.

[21] R. A. Bjork, "Memory and metamemory considerations in the training of human beings," in Metacognition: Knowing About Knowing, J. Metcalfe and A. Shimamura, Eds. MIT Press, pp. 185–205, 1994.

[22] K. Loibl, R. Roll, and N. Rummel, "Towards a theory of when and how problem solving followed by instruction supports learning," Educational Psychology Review, vol. 29, no. 4, pp. 693–715, 2017.

[23] K. Prathyusha, T. Maha Lakshmi, and P. S. V. Vachaspati, "Real-time multilingual traffic sign detection and alert system using YOLOv8," International Journal of Innovative Research in Technology, vol. 12, no. 8, pp. 45–53, 2026.

[24] L. S. Vygotsky, Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, Cambridge, MA, 1978.

[25] P. Denny, V. Kumar, and N. Giacaman, "Conversing with GPT-3 to improve code completion and programming student learning outcomes," in Proc. ACM Technical Symposium on Computer Science Education (SIGCSE), pp. 1–7, 2023.