CODE-GEN: A RAG-Based Agentic AI System for Multiple-Choice Question Generation

Xiaojing Duan[1 [0000−0003−0822−2022]], Frederick Nwanganga[1 [0000−0002−4514−6527]], and Chaoli Wang[1 [0000−0002−0859−3619]]

University of Notre Dame, Notre Dame IN 46556, USA
{xduan,fnwangan,chaoli.wang}@nd.edu

Abstract. We present CODE-GEN, a retrieval-augmented generation (RAG)-based agentic AI system for generating context-aligned multiple-choice questions to develop learners' code comprehension. CODE-GEN employs a dual-agent architecture in which a Generator agent produces questions aligned with course-specific learning objectives and a Validator agent independently assesses quality across seven pedagogical dimensions, both augmented with specialized tools for computational accuracy and code verification. To evaluate CODE-GEN, six subject-matter experts (SMEs) judged 288 AI-generated questions, producing 2,016 human-AI rating pairs and 131 instances of qualitative feedback. Results show strong system performance, with human-validated success rates ranging from 79.9% to 98.6%. CODE-GEN achieves high reliability on dimensions suited to computational verification and explicit criteria matching, including question clarity, code validity, concept alignment, and correct answer validity. In contrast, human expertise remains essential for dimensions requiring deeper instructional judgment, such as distractor design and feedback quality. These findings provide evidence-based design guidelines for the strategic allocation of human and AI effort in AI-assisted educational content generation.

Keywords: LLMs · Retrieval-augmented generation · Agentic AI · Multi-agent architecture · Human evaluation · Pedagogical alignment

1 Introduction

Recent advances in large language models (LLMs) have generated substantial interest in their potential to support teaching and learning in educational contexts [4, 15]. While prior studies demonstrate that LLMs can generate a wide range of educational artifacts [7, 1], their adoption in authentic classroom settings remains limited [10, 12]. This limitation stems from two persistent challenges. First, content produced by general-purpose LLMs is often overly generic and insufficiently aligned with course-specific learning objectives [8, 14], undermining instructional validity and reducing instructor trust in AI-assisted content generation. Second, although recent work in agentic AI has introduced multi-agent systems with specialized roles for generation, critique, and revision to improve content quality [18,

9], these systems frequently assume that automated critique agents provide reliable evaluations. In practice, such assumptions are problematic, as LLM-based evaluators are known to exhibit hallucinations, bias, and inconsistent judgment [16, 11]. Without systematic validation against human expert judgment, errors introduced at the evaluation stage risk being amplified rather than corrected.

To address these challenges, we present CODE-GEN (Context-aligned, Output-validated, Dual-agent, Expert-guided GENeration), an agentic AI system for generating contextually grounded multiple-choice coding comprehension questions. CODE-GEN integrates retrieval-augmented generation (RAG) with a dual-agent architecture in which a Generator agent produces multiple-choice coding questions aligned with course-specific learning objectives, and a Validator agent independently assesses question quality across seven pedagogical dimensions derived from established multiple-choice item-writing guidelines. Both agents are augmented with specialized tools to support computational accuracy and reliable code execution verification. To evaluate the effectiveness of this agentic approach, we conducted a comprehensive human evaluation study in which six subject-matter experts (SMEs) judged 288 AI-generated questions, yielding 2,016 human-AI judgment pairs and 131 instances of qualitative feedback. Analysis of these SME judgments provides empirical evidence of where agentic AI systems can reliably support educational content generation and where human expertise remains essential, thereby characterizing both the strengths and limitations of LLM-based approaches to AI-assisted educational content generation.

The main contributions of the study include: (1) We introduce CODE-GEN, an agentic AI system that generates context-aligned multiple-choice questions validated against explicit pedagogical criteria. (2) Unlike prior multi-agent systems that assume automated critique agents are reliable, we treat automated assessment as an empirical object of study, systematically comparing Validator assessments against SME judgments across seven pedagogical dimensions to identify where automated assessment succeeds and where it falls short. (3) We derive evidence-based design guidelines demonstrating that AI can provide scalable, first-line quality control for dimensions involving computational verification and explicit criteria matching, while human experts remain essential for pedagogical judgment, including plausible distractor design and instructionally rich feedback.

2 Method

2.1 CODE-GEN

CODE-GEN is a RAG-based agentic AI system for generating and validating multiple-choice coding comprehension questions that are closely aligned with course-specific learning objectives. Figure 1 illustrates the end-to-end workflow for CODE-GEN. The workflow begins with instructors uploading instructional materials, including learning objectives and example questions. These materials define the authoritative pedagogical context and are indexed through a RAG

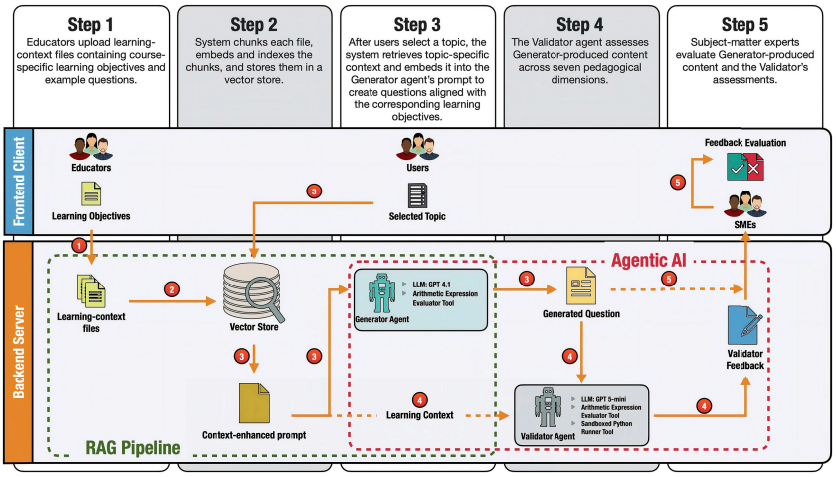

Fig. 1. CODE-GEN end-to-end system workflow.

pipeline. When a user selects a topic, relevant instructional examples are retrieved and provided to a Generator agent, which produces multiple-choice questions grounded in the provided context. The Validator agent then independently aesses each question item across seven pedagogical dimensions as described in Table 1. The dimensions are: question stem clarity [6], code validity [3], concept alignment [2], correct answer validity [17], distractor quality [5], correct answer feedback quality [6], and distractor feedback quality [5].

Table 1. Evaluation dimensions and Validator assessment outputs.

Evaluation Dimension	Description	Classification Output	Rationale Output
Question Stem Clarity	Clarity of the question stem	Yes/No	Why the question stem is or is not clear
Code Validity	Validity of the generated code	Yes/No	Why the code is or is not valid
Concept Alignment	Alignment of the generated question with the provided context	Yes/No	Why the question does or does not align with the provided context
Correct Answer Validity	Validity of the marked correct answer	Yes/No	Why the marked correct answer is or is not valid
Distractor Quality	Plausibility and pedagogical value of incorrect options	Good/Poor	Why the distractor quality is good or poor
Correct Answer Feedback Quality	Quality of feedback in explaining why the answer is correct and reinforcing the underlying concept	Good/Poor	Why is the correct answer feedback quality good or poor
Distractor Feedback Quality	Quality of feedback in explaining why the distractors are incorrect	Good/Poor	Why the distractor feedback quality is good or poor

RAG Pipeline To ensure the generated questions are contextually aligned with course-specific learning objectives, CODE-GEN employs a RAG pipeline

tailored for coding comprehension multiple-choice questions. Uploaded instructional materials are segmented using a domain-specific chunking strategy that preserves semantic coherence. Each semantically coherent chunk is embedded using the OpenAI text-embedding-3-small model and indexed in a Facebook AI Similarity Search (FAISS) vector store. When a user initiates question generation for a given topic, the system identifies the most relevant instructional examples through nearest-neighbor retrieval. These retrieved examples are embedded into the Generator agent's prompt, constraining generation to the intended instructional context.

Model Selection To select models for the Generator and Validator agents, we conducted comparative experiments across state-of-the-art commercial LLMs at the time of the study (Claude Sonnet 4.5, Gemini 2.5 Pro, GPT-5-mini, and GPT-4.1), prioritizing API-accessible options to ensure reproducibility and practical deployability. Generator selection criteria emphasized novelty relative to retrieved context, answer correctness, and response latency; GPT-4.1 was selected for its balance across all three dimensions. Validator selection focused on detecting common quality issues such as incorrect answer keys and answer-feedback inconsistencies; GPT-5-mini was selected for its reliable error detection and concise validation output.

Tool Augmentation Augmenting AI agents with external tools is an established strategy for extending capabilities beyond pure language generation [13]. In CODE-GEN, we adopt this approach to address two well-documented LLM limitations. First, because LLMs frequently miscalculate multi-step arithmetic expressions involving operator precedence, we developed an Arithmetic Expression Evaluator to support the Generator and Validator with deterministic arithmetic computation and verification. Second, to support automated validation of code execution results, we developed a Sandboxed Python Runner for the Validator to execute code in a restricted environment, capture outputs, and inspect variable states. Together, these tools improve quality assurance while maintaining operational security.

Prompt Engineering The Generator and Validator agents use structured prompts designed to ensure pedagogical alignment, reliable tool usage, and strict output formatting for consistent parsing and scalable downstream analysis. The complete prompts are available on GitHub.[1]

2.2 Human Subject-Matter Expert Evaluation

CODE-GEN leverages human expertise to verify the Validator's assessments and mitigate the risk of propagating incorrect or misleading feedback during iterative refinement cycles. We recruited six SMEs (three men and three women), all with extensive experience teaching introductory programming. Following an orientation session to establish a shared understanding of evaluation criteria, SMEs evaluated 288 questions and the associated validations using CODE-GEN.

[1] https://github.com/XiaojingDuan/CODE-GEN/.

Figure 2 illustrates an example of the evaluation interface using a while-loop-related question created by the Generator. Adjacent to the question, the user interface displays the Validator's analysis across seven evaluation dimensions. For each question, SMEs judged the generated items alongside the Validator's

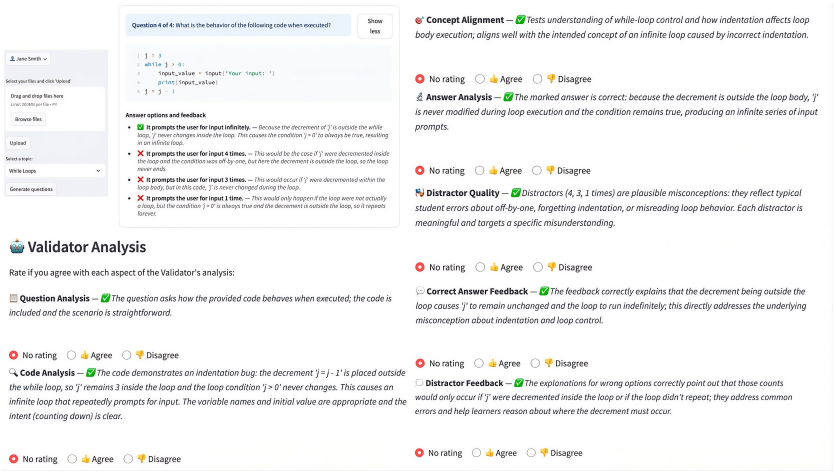

Fig. 2. CODE-GEN web user interface.

assessments, indicated agreement or disagreement for each dimension using the explicit Agree or Disagree button, and provided textual rationales when they disagreed. Across all experts and dimensions, this process yielded 2,016 SME-Validator rating pairs and 131 instances of qualitative feedback. Because CODE-GEN generates unique questions for each user, SMEs evaluated distinct question sets. As such, traditional inter-rater reliability measures such as Cohen's κ are not applicable, as they require multiple raters judging the same items. To assess evaluator consistency, we examined the SME-Validator agreement rates across all SMEs. Results show consistent agreement rates across all dimensions (82.5%–98.4%), with five of seven exhibiting standard deviations of 3.8% or below, indicating that SMEs applied comparable evaluation standards despite reviewing different question sets.

3 Results and Discussion

3.1 System-Level Quality Analysis Results

To evaluate CODE-GEN's performance, we treated human SMEs' judgments as ground truth and the Validator's assessments as predictions in a binary classification framework. Each SME-Validator rating pair was assigned to one of four outcome categories: **Success (TP)**, representing high-quality content correctly

accepted; **Failure (FP)**, representing low-quality content incorrectly accepted; **Safeguarding (TN)**, representing low-quality content correctly rejected; and **Inefficiency (FN)**, representing high-quality content incorrectly rejected. Figure 3 shows system performance by outcome category across all seven evaluation dimensions.

Overall, CODE-GEN performed strongly. Five of the seven dimensions achieved human-validated success rates of 92% or higher. Concept alignment exhibited the highest success rate (98.6%), with only 0.3% failure. This exceptional performance validates that the RAG-based agentic design enables CODE-GEN to produce highly targeted questions. The other dimensions grounded in explicit

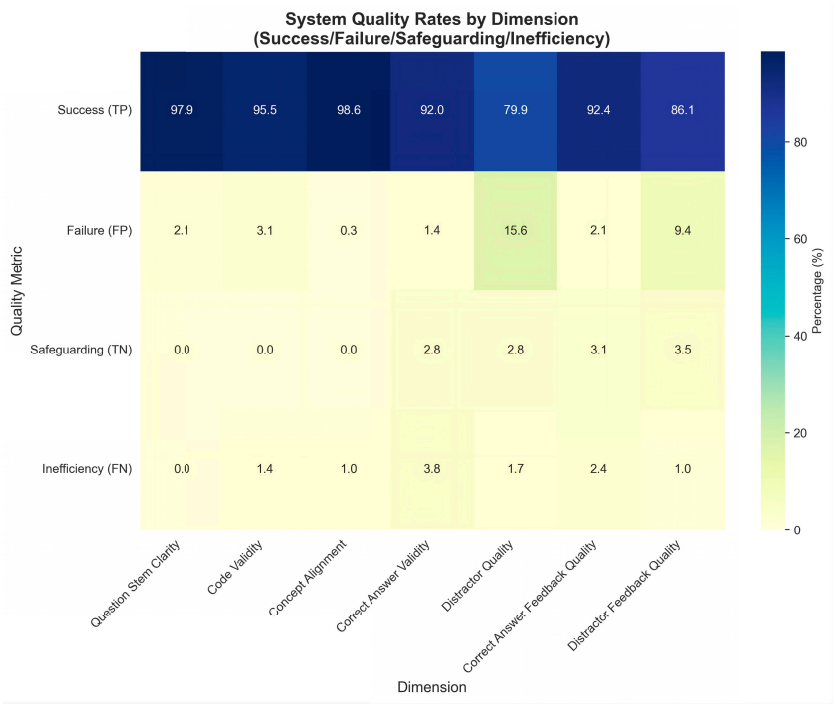

Fig. 3. System quality by evaluation dimension.

criteria and tool-augmented verification, including question stem clarity, code validity, correct answer validity, and correct answer feedback quality, consistently achieved high success rates. In contrast, dimensions that require nuanced pedagogical judgment achieved comparatively lower success rates. Distractor quality yielded the lowest success rate (79.9%) and the highest failure rate (15.6%), while distractor feedback quality showed a success rate of 86.1% with a comparatively elevated failure rate (9.4%). These findings reinforce the continued need for human oversight in evaluating whether distractors effectively target com-

mon misconceptions and whether feedback adequately elaborates on underlying concepts.

3.2 System-Level Limitation Analysis Results

To better understand the limitations of CODE-GEN, we analyzed qualitative feedback from SMEs in the FP and FN cases. FP cases predominantly reflected limitations in pedagogical judgment rather than technical correctness. In the distractor quality dimension, the Generator often produced distractors that were syntactically valid and superficially plausible but failed to target common misconceptions. SMEs consistently noted that higher-quality distractors would better reflect known novice confusions. For example, when the Validator approved low-quality distractors, one SME critiqued: *"These distractors are obviously wrong. Better examples include pop('Leprechaun'), remove(4), and remove(3). They reflect students' common confusion with whether to use index position or values in .pop() vs. .remove() and miscount the index position by 1."* Similarly, FP cases in feedback-related dimensions showed that the Validator approved explanations that correctly described surface-level mechanics but lacked deeper pedagogical elaboration. For example, when the Validator approved low-quality feedback, an SME noted: *"The feedback didn't explain the given while loop reiterates after printing 'stop' because it reaches the end of the current iteration."*

FN cases exposed structural weaknesses in the Validator's reasoning process. A common pattern involved misinterpretation of answer schemas, where the Validator correctly analyzed the code output but confused the answer value with the option position. For example, when the correct answer was 2, the Validator correctly derived 2 but identified the 2nd option as marked correct even though the 1st option was the designated correct answer. An SME noted *"The marked answer is the value 2 (option 1), not option 2 (value 3)."* Another recurring issue was internal inconsistency, in which the Validator's detailed textual analysis affirmed alignment or correctness while its final binary classification indicated the opposite. For example, the Validator's analysis explicitly affirmed concept alignment yet generated negative classifications. An SME commented, *"The question aligns with the concept as stated above, but the final evaluation shows disagreement, which doesn't make any sense."*

3.3 Discussion

The empirical results show that CODE-GEN achieves strong human-validated success on dimensions grounded in explicit criteria and computational verification, with success rates ranging from 79.9% to 98.6%. In particular, the RAG pipeline played a central role in ensuring alignment between generated questions and course-specific learning objectives, as reflected in the exceptionally high concept alignment success rate. The agentic separation between generation and validation enabled systematic, dimension-level quality control, while tool augmentation substantially improved reliability on technically grounded dimensions, such as code validity, correct-answer validity, and correct-answer feedback

quality. However, the analysis of human-AI disagreement cases shows automated validation was prone to approving technically correct but instructionally shallow items, underscoring the need for human oversight on pedagogically sensitive dimensions such as distractor design and feedback quality. Taken together, these findings offer evidence-based design guidelines for AI-assisted educational content generation: AI can provide scalable first-line quality control while human experts remain essential for ensuring pedagogical depth and overall instructional quality.

4 Limitations and Future Work

While this study focused on introductory Python, the underlying architecture, including the dual-agent workflow, RAG-based retrieval, and a dimension-level evaluation framework, is domain-agnostic. Future work can extend CODE-GEN to other domains by adapting domain-specific components such as the chunking strategy and tool augmentation. Another important future direction is iterative refinement. CODE-GEN currently uses single-pass generation by design, as iterative refinement requires confidence in the Validator's reliability, which this study was designed to establish. The findings now provide an empirical basis for future selective iterative refinement.

Acknowledgments. We gratefully acknowledge the six subject matter experts whose rigorous evaluations provided the empirical foundation for this research.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Barros, J., Moraes, L.O., Oliveira, F., Delgado, C.A.D.M.: Large Language Models Generating Feedback for Students of Introductory Programming Courses. In: Cristea, A.I., Walker, E., Lu, Y., Santos, O.C., Isotani, S. (eds.) Artificial Intelligence in Education, vol. 15878, pp. 421–433. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-98417-4_30
2. Butler, A.C.: Multiple-choice testing in education: Are the best practices for assessment also good for learning? Journal of Applied Research in Memory and Cognition **7**(3), 323–331 (Sep 2018). https://doi.org/10.1016/j.jarmac.2018.07.002
3. Doughty, J., Wan, Z., Bompelli, A., Qayum, J., Wang, T., Zhang, J., Zheng, Y., Doyle, A., Sridhar, P., Agarwal, A., Bogart, C., Keylor, E., Kultur, C., Savelka, J., Sakr, M.: A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education. In: Proceedings of the 26th Australasian Computing Education Conference. pp. 114–123. ACM, Sydney NSW Australia (Jan 2024). https://doi.org/10.1145/3636243.3636256
4. Dutulescu, A., Ruseti, S., Iorga, D., Dascalu, M., McNamara, D.S.: YMCQ: Reasoning-Enhanced MCQ Generation. In: Cristea, A.I., Walker, E., Lu, Y., Santos, O.C., Isotani, S. (eds.) Artificial Intelligence in Education, vol. 15882, pp. 308–315. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-98465-5_39

5. Gierl, M.J., Bulut, O., Guo, Q., Zhang, X.: Developing, Analyzing, and Using Distractors for Multiple-Choice Tests in Education: A Comprehensive Review. Review of Educational Research **87**(6), 1082–1116 (Dec 2017). https://doi.org/10.3102/0034654317726529
6. Haladyna, T.M., Downing, S.M., Rodriguez, M.C.: A Review of Multiple-Choice Item-Writing Guidelines for Classroom Assessment. Applied Measurement in Education **15**(3), 309–333 (Jul 2002). https://doi.org/10.1207/S15324818AME1503_5
7. Hoq, M., Vandenberg, J., Jiao, S., Lee, S., Mott, B., Norouzi, N., Lester, J., Akram, B.: Facilitating Instructors-LLM Collaboration for Problem Design in Introductory Programming Classrooms (May 2025). https://doi.org/10.48550/arXiv.2504.01259
8. Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., Stadler, M., Weller, J., Kuhn, J., Kasneci, G.: ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences **103**, 102274 (Apr 2023). https://doi.org/10.1016/j.lindif.2023.102274
9. Kostopoulos, G., Gkamas, V., Rigou, M., Kotsiantis, S.: Agentic AI in Education: State of the Art and Future Directions. IEEE Access **13**, 177467–177491 (2025). https://doi.org/10.1109/ACCESS.2025.3620473
10. Kurdi, G., Leo, J., Parsia, B., Sattler, U., Al-Emari, S.: A Systematic Review of Automatic Question Generation for Educational Purposes. International Journal of Artificial Intelligence in Education **30**(1), 121–204 (Mar 2020). https://doi.org/10.1007/s40593-019-00186-y
11. Lin, X., Ning, Y., Zhang, J., Dong, Y., Liu, Y., Wu, Y., Qi, X., Sun, N., Shang, Y., Wang, K., Cao, P., Wang, Q., Zou, L., Chen, X., Zhou, C., Wu, J., Zhang, P., Wen, Q., Pan, S., Wang, B., Cao, Y., Chen, K., Hu, S., Guo, L.: LLM-based Agents Suffer from Hallucinations: A Survey of Taxonomy, Methods, and Directions (2025). https://doi.org/10.48550/ARXIV.2509.18970
12. Lyu, W., Wang, Y., Chung, T.R., Sun, Y., Zhang, Y.: Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study. In: Proceedings of the Eleventh ACM Conference on Learning @ Scale. pp. 63–74. ACM, Atlanta GA USA (Jul 2024). https://doi.org/10.1145/3657604.3662036
13. Masterman, T., Besen, S., Sawtell, M., Chao, A.: The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey (2024). https://doi.org/10.48550/ARXIV.2404.11584
14. Memarian, B., Doleck, T.: ChatGPT in education: Methods, potentials, and limitations. Computers in Human Behavior: Artificial Humans **1**(2), 100022 (Aug 2023). https://doi.org/10.1016/j.chbah.2023.100022
15. Scaria, N., Chenna, S.D., Subramani, D.: Automated Educational Question Generation at Different Bloom's Skill Levels using Large Language Models: Strategies and Evaluation (2024). https://doi.org/10.48550/ARXIV.2408.04394
16. Stureborg, R., Alikaniotis, D., Suhara, Y.: Large Language Models are Inconsistent and Biased Evaluators (2024). https://doi.org/10.48550/ARXIV.2405.01724
17. Towns, M.H.: Guide To Developing High-Quality, Reliable, and Valid Multiple-Choice Assessments. Journal of Chemical Education **91**(9), 1426–1431 (Sep 2014). https://doi.org/10.1021/ed500076x
18. Yuksel, K.A., Sawaf, H.: A Multi-AI Agent System for Autonomous Optimization of Agentic AI Solutions via Iterative Refinement and LLM-Driven Feedback Loops (Dec 2024). https://doi.org/10.48550/arXiv.2412.17149