ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

Enhancing Programming Education using GPT-Based Code Review and Automatic Comment Generation

Shaik Ayesha[1], Manikonda Ramya Krishna[2], Sevugarajan.S[3], Sateesh Yalavarthi[4], Donepudi Prasanthi[5]

#1 Post Graduate Scholar, Department of CSE, Vikas College of Engineering & Technology, Nunna, Andhra Pradesh, India- 521212.

#2 Assistant Professor, Department of CSE, Vikas College of Engineering & Technology, Nunna, Andhra Pradesh, India- 521212.

#3 Professor, Department of ECE, Vikas College of Engineering & Technology, Nunna, Andhra Pradesh, India- 521212.

#4 Post Graduate Scholar, Department of CSE, Vikas College of Engineering & Technology, Nunna, Andhra Pradesh, India- 521212.

#5 Post Graduate Scholar, Department of CSE, Vikas College of Engineering & Technology, Nunna, Andhra Pradesh, India- 521212.

Mail Id: ayeshamca333@gmail.com[1], pranavsourya079@gmail.com [2], dean_academics@vikasinstitutionsnunna.org[3] , sateeshthermal@gmail.com [4], donepudi.prasanthi@gmail.com [5]

Abstract: With the rapid growth of online and self-paced programming education, providing accurate and timely feedback to learners has become a significant challenge. Traditional automated evaluation systems primarily rely on test-case execution and fail to deliver meaningful, explanatory feedback, while existing Large Language Model (LLM)-based tools often generate complete solutions, raising concerns regarding academic integrity and inefficient resource usage. To address these limitations, this paper proposes an intelligent GPT-4o–based Code Review System designed to generate precise, feedback-oriented responses without revealing full solutions. The system incorporates a Code Review Module with a Review Necessity Chain and Code Correctness Check to focus evaluation on

relevant code segments and reduce unnecessary computation.

In addition, an NLP-based automatic comment generation module is introduced as an extension to enhance code readability by generating descriptive inline comments from source code structure and identifiers. The proposed system is evaluated using BERT-Score to measure semantic similarity between generated and reference review comments, achieving an F1-score of 88%, indicating strong alignment with expert feedback. A Flask-based web interface enables real-time interaction, allowing users to upload code and receive structured review comments along with automatically generated explanations. The results demonstrate that the proposed framework improves code comprehension, supports independent learning, and provides a scalable and educationally effective solution for automated programming feedback systems.

Index terms *- — GPT-4o, automated code review, natural language processing, automatic comment generation, programming education, large language models, educational feedback, code comprehension, BERT-Score, software learning systems*

1. INTRODUCTION

The rapid advancement of digital technologies has significantly transformed modern education, making programming and computational thinking essential skills for learners across all levels. With the widespread adoption of online and self-paced learning platforms, students are increasingly required to develop coding skills independently. However, one of the major challenges in programming education is the lack of timely, accurate, and meaningful feedback on code submissions. Traditional approaches rely heavily on manual evaluation by instructors, which is time-consuming, inconsistent, and difficult to scale for large numbers of learners.

To address these limitations, automated assessment systems and Large Language Models (LLMs) have been introduced to support programming education. Existing systems primarily rely on test-case execution to evaluate correctness, providing feedback in the form of compiler errors or failed test cases. While such approaches are effective in identifying errors, they often fail to explain the underlying issues in a way that is understandable for novice learners. Recent advancements in LLMs have enabled the generation of natural language feedback, offering more readable and context-aware explanations. However, many existing LLM-based tools tend to generate complete code solutions, which can negatively impact learning outcomes and raise concerns regarding academic integrity. Additionally, these systems may consume unnecessary computational resources when processing incorrect or incomplete code.

Accurate and educational feedback is crucial for improving learners' problem-solving skills, code quality, and understanding of programming concepts. Code review plays a vital role in this process by providing suggestions for optimization, identifying logical errors, and promoting best coding practices. Despite its importance, there is still a lack of intelligent systems that can deliver precise, context-aware feedback while avoiding full solution disclosure. Furthermore, many learners struggle with understanding code due to the absence of proper

documentation or comments, which affects code readability and maintainability.

To overcome these challenges, this paper proposes an intelligent GPT-4o–based code review system that focuses on generating structured and feedback-oriented responses rather than complete solutions. The system integrates a Code Review Module with a Review Necessity Chain and Code Correctness Check to ensure efficient and relevant analysis of submitted code. In addition, an NLP-based automatic comment generation module is introduced as an extension to enhance code comprehension by generating descriptive inline comments from code structure and identifiers. The proposed system is evaluated using BERT-Score to measure semantic similarity between generated and reference review comments, demonstrating strong performance with an F1-score of 88%.

The main contributions of this work include the design of a learning-oriented code review framework, the integration of automatic comment generation to improve readability, and the development of a scalable web-based system for real-time feedback. Overall, the proposed approach aims to enhance programming education by providing accurate, efficient, and ethically responsible automated feedback.

2. LITERATURE SURVEY

a) Feedback-Generation for Programming Exercises With GPT-4:

Since the widespread availability of Large Language Models (LLMs) and associated applications, a number of research have looked into how they could help teachers and students in higher education. In the context of large programming courses, where students can benefit from feedback and pointers if given promptly and at scale, LLMs like Codex, GPT-3.5, and GPT 4 have demonstrated encouraging outcomes. The quality of the output produced by GPT-4 Turbo for prompts that include both the programming task definition and a student's submission as input is examined in this research. GPT-4 was requested to provide comments on 55 randomly selected, real student programming submissions, and two assignments from a basic programming course were chosen. In terms of accuracy, customization, fault localization, and other characteristics found in the content, the output was qualitatively examined. GPT-4 Turbo exhibits significant improvements over previous studies and analysis of GPT-3.5. The output, for instance, is more consistent and organized. Additionally, GPT-4 Turbo can reliably detect incorrect casing in the output of student applications. In certain instances, the student program's output is also included in the feedback. Inconsistent feedback was also seen, such as saying that the submission is correct but that an issue has to be addressed. The current study advances our knowledge of the possibilities and constraints of LLMs as well as how to incorporate them into instructional situations, e-assessment systems, and teaching students utilizing GPT-4-based apps.

b) Effective Teaching through Code Reviews: Patterns and Anti-patterns:

A common and crucial step in the software development process is code reviews. In contrast to something more detached and formal, like a class, they also provide a special, at-scale opportunity for training developers inside the framework of their daily development operations. However, research on

successful education through code reviews—which emphasize learning for the author rather than only code modifications—is lacking. We fill this vacuum with a case study at Google, where 12 patterns and 15 anti-patterns in code reviews that affect learning were found through interviews with 14 developers. For example, learning is facilitated by explanatory reasoning, standards-based sample answers, and a constructive tone; learning is hampered by harsh remarks, excessively superficial criticism, and non-pragmatic reviewing that disregards authors' limitations. Through member checks, reviewer interviews, a literature study, and a poll of 324 developers, we verified our qualitative results. The influence of social dynamics in code reviews on learning is empirically demonstrated by this extensive study. We provide useful suggestions on how to structure constructive evaluations to foster a positive learning environment based on our findings.

c) Improving the Coverage of GPT for Automated Feedback on High School Programming Assignments:

For inexperienced programmers, feedback on bad code is crucial. In the past, Automated Program Repair (APR) technologies have been used to produce feedback for errors in beginning programming courses. Due to their demonstrated ability to produce both human-readable text and code, Large Language Models (LLMs) have become a compelling alternative to automatic feedback production. In this paper, we evaluate both APR and LLMs on a diverse dataset that includes 366 incorrect submissions for a set of 69 problems with varying complexity from a public high school in order to compare the efficacy of LLMsto APR techniques for code repair and feedback generation in the context of high school Python programming assignments. We demonstrate that, given a suitable assessment oracle, LLMs outperform APR approaches in repair generation. The direct invocation of such LLMs nevertheless has several drawbacks, even if the most advanced GPTs may typically produce feedback for defective code. Specifically, GPT-4 may miss up to 16% of the bugs, provide erroneous feedback around 8% of the time, and experience hallucinations approximately 5% of the time. We demonstrate how a novel design that uses a conversational interactive loop to invoke GPT may increase GPT-3.5T's repair coverage from 64.8% to 74.9%, matching the capabilities of the cutting-edge LLM GPT-4. Similarly, using the same methods, the coverage of GPT-4 may be increased from 74.9% to 88.5% in just five rounds.

d) Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments:

Large language models (LLMs), which can produce text, graphics, and source code in response to human requests, are the result of recent developments in artificial intelligence. In this work, we investigate the potential of an LLM—OpenAI's GPT-3 model—to offer comments on student-written code. In particular, we investigate the viability of using GPT-3 to review, evaluate, and recommend modifications to code provided by students in an online programming test for an undergraduate Python programming course. We gathered 1211 student code submissions from seven programming test problems, and we gave the GPT-3 model distinct prompts to review, evaluate, and offer recommendations on these submissions. We discovered that the accuracy of the model's feedback for student inputs varied greatly.

Between 57% and 79% of respondents correctly checked that the code was correct, between 41% and 77% correctly critiqued the code, and between 32% and 93% correctly suggested modifications to the code. Additionally, we discovered situations in which the model produced inconsistent and inaccurate input. These results imply that it is presently not possible to "directly" employ models such as GPT-3 to provide students feedback on programming tests.

e) Enhancing Student Focus and Problem-Solving with Real-Time LLM Feedback on Compiler Errors:

Because programming syntax is so complicated, automating compiler error reporting in educational contexts has always been challenging. Although the usefulness of this feedback is yet unknown, large language models (LLMs) show promise for solving this problem at scale by offering customized feedback based on individual code inputs. In a randomized controlled trial, this study assessed how well GPT-4o produced real-time feedback for compiler faults. An automated programming evaluation platform received 22,674 bits of code from 248 CS1 students. While the Control group did not get LLM comments, students in the Experimental group did. The findings revealed that students who got LLM feedback submitted less attempts at non-compiling code and gave it a high utility rating. When compared to the Control group, these pupils also performed noticeably better when it came to fixing mistakes in successive tries. The LLM feedback group self-reported increased attention and decreased levels of "confrustion" (a mix of perplexity and frustration) following compiler failures, according to affective questionnaires. Students in the Experimental group solved programming tasks faster and showed a notable improvement in fixing errors across tries when LLM feedback was momentarily turned off. On a simulated test, however, there were no discernible variations between the groups' final results. These results imply that while LLM-generated feedback may not result in improved end performance, it can enhance students' coding experience, engagement, and problem-solving effectiveness in the early stages of computer science instruction.

3. METHODOLOGY

i) Proposed Work:

The proposed work introduces an intelligent GPT-4o–based Code Review System designed to provide accurate, context-aware, and educational feedback for programming learners without revealing complete solutions. The system processes learner-submitted source code through a structured pipeline that includes data preprocessing, Review Necessity Chain (RNC), and Code Correctness Check Module (CCM). These components ensure that only relevant code segments are analyzed, reducing unnecessary computation while maintaining feedback precision. The GPT-based model then generates constructive review comments focusing on error identification, code quality improvement, and adherence to best practices, thereby supporting independent learning and preserving academic integrity.

In addition, the system incorporates an NLP-based automatic comment generation module as an extension to enhance code readability and comprehension. This module analyzes code structure, including functions, classes, variables, and control-flow patterns, to automatically generate meaningful inline comments for undocumented code. The system

is implemented using a Flask-based web interface, allowing users to upload code files and receive structured review feedback along with auto-generated comments in real time. The overall framework is evaluated using BERT-Score, achieving an F1-score of 88%, demonstrating strong alignment with expert feedback and confirming the effectiveness of the proposed approach in programming education environments.

ii) System Architecture:

The system architecture of the proposed GPT-based code review system is designed to process learner-submitted code through multiple validation and feedback modules to generate accurate and educational responses. Initially, the user submits source code to the system, which is first passed through the Code Validation Module to check for basic syntax and structural correctness. If the code passes validation, it is forwarded to the Code Review Module, which consists of the Review Necessity Chain and Review Comment Generating Chain. These components analyze the code to identify relevant sections requiring review and generate structured feedback comments focused on improvement and best practices.

Simultaneously, the submitted code is also processed by the Code Correctness Check Module, which includes the Answer Check Component and Strict Code Check Chain to evaluate logical correctness and validate outputs. The system then combines the results from both modules and provides responses to the user in the form of review comments and correctness feedback, such as correct, incorrect, or requiring further refinement. All interactions and outputs are stored in the user log database for tracking and future analysis. This architecture ensures efficient, accurate, and context-aware feedback while maintaining a clear separation between code review and correctness evaluation processes.

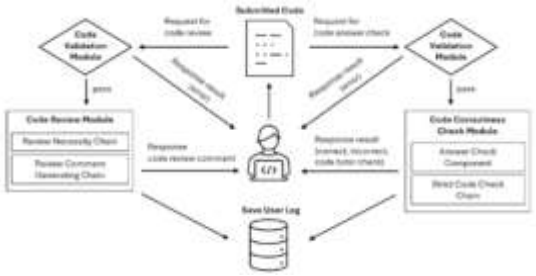

Fig1 proposed architecture

iii) Modules:

Code Submission Module

- Accepts learner source code through web interface
- Supports file upload and direct code input
- Sends code to processing pipeline

Code Validation Module

- Checks syntax and basic structure of code
- Filters invalid or incomplete submissions
- Ensures only valid code proceeds further

Code Review Module

- Analyzes code for quality and best practices
- Uses Review Necessity Chain to focus important parts
- Generates structured review comments

Code Correctness Check Module

- Evaluates logical correctness of code
- Uses answer checking and strict validation

- Provides result as correct, incorrect, or needs improvement

NLP-Based Automatic Comment Generation Module (Extension)

- Generates inline comments for undocumented code
- Uses function names, variables, and structure
- Improves readability and understanding

Ask Code Tutor Module

- Provides additional explanations and guidance
- Helps learners understand mistakes
- Offers improvement suggestions without full solutions

Evaluation Module

- Measures performance using BERT-Score
- Compares generated feedback with reference comments
- Ensures semantic accuracy and quality

User Log and Storage Module

- Stores user inputs and generated outputs
- Maintains history of code reviews
- Supports future analysis and system improvement

iv) Algorithms:

1: GPT-Based Code Review Generation

The GPT-based code review generation algorithm begins by accepting learner-submitted source code through the system interface. The input code is preprocessed by performing parsing, cleaning, and tokenization to extract important elements such as functions, classes, variables, and control-flow structures. The Review Necessity Chain (RNC) is then applied to identify relevant portions of the code that require evaluation, ensuring efficient processing. This is followed by code validation and correctness checking to verify logical and syntactic consistency. The refined code is then passed to the GPT-4o model with carefully designed prompts that restrict output to feedback-only responses. The model generates structured and constructive review comments focusing on error identification, code optimization, and adherence to best practices, without revealing complete solutions. Finally, the generated feedback is delivered to the user and stored in the system for future analysis.

2: BERT-Score Evaluation Algorithm

The BERT-Score evaluation algorithm is used to measure the semantic similarity between model-generated code review comments and instructor-provided reference feedback. Initially, both the generated comments and reference comments are collected and preprocessed. These textual inputs are then converted into contextual embeddings using a pre-trained BERT model, which captures the semantic meaning of each token. The algorithm computes similarity scores by comparing corresponding tokens in both embeddings, resulting in precision, recall, and F1-score values. These metrics indicate how closely the generated feedback aligns with expert-written comments in terms of meaning rather than exact wording. In the proposed

system, the BERT-Score achieves an F1-score of approximately 88%, demonstrating strong performance and confirming the effectiveness of the feedback generation process.

4. EXPERIMENTAL RESULTS

The proposed GPT-based code review system was implemented and evaluated using a combination of Jupyter Notebook for model development and a Flask-based web application for real-time user interaction. The system was trained and tested using a publicly available code review dataset, where each sample contains source code and corresponding expert review comments. During experimentation, learner-submitted code without comments was provided as input, and the system successfully generated structured review feedback along with automatic inline comments through the NLP-based extension module.

The results demonstrate that the system effectively identifies code issues such as naming inconsistencies, logical errors, and improvement suggestions without generating complete solutions. The Flask interface provides a user-friendly environment where input code, generated review comments, and auto-generated annotations are displayed side-by-side, enabling easy comparison and understanding. The extension module further enhances the output by automatically inserting descriptive comments for functions, variables, and control-flow structures, improving code readability and comprehension.

For quantitative evaluation, the performance of the proposed model was measured using BERT-Score, which evaluates semantic similarity between generated and reference review comments. The system achieved an F1-score of 88%, indicating strong alignment with instructor-provided feedback. This confirms that the generated comments are not only grammatically correct but also semantically meaningful and educationally useful. Overall, the experimental results validate that the proposed system provides accurate, efficient, and scalable feedback, making it suitable for real-world programming education environments.

Accuracy: A test's accuracy is its capacity to distinguish healthy from ill cases. Find the percentage of instances with genuine positives and negatives to assess test accuracy.

Accuracy = TP + TN /(TP + TN + FP + FN)

$$Accuracy = \frac{(TN + TP)}{T}$$

Precision: Classification accuracy or positive cases constitute precision. The formula for accuracy is:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

$$\Pr e\,cision = \frac{TP}{(TP + FP)}$$

Recall: A model's recall measures its ability to recognize all appropriate machine learning class instances. The ratio of accurately predicted positive observations to total positives indicates a model's class instance detection skill.

$$Recall = \frac{TP}{(FN + TP)}$$

mAP: Mean Average Precision ranks quality. It considers the number and order of relevant ideas.

Calculating MAP at K uses the arithmetic mean of each user or query's Average Precision (AP).

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

$$AP_k = \text{the AP of class } k$$

$$n = \text{the number of classes}$$

F1-Score: A high F1 score suggests an accurate machine learning model. Integrating recall and precision improves model correctness. Accuracy measures how often a model predicts a dataset correctly.

$$F1 = 2 \cdot \frac{(Recall \cdot \Pr e\,cision)}{(Recall + \Pr e\,cision)}$$

Fig2 Uploaded code

Fig3 Results

5. CONCLUSION

This paper presented an intelligent GPT-4o–based code review system that delivers accurate, context-aware, and educational feedback for programming learners without revealing complete solutions. By integrating a Code Review Module with a Review Necessity Chain and Code Correctness Check, the system focuses on relevant code segments, reduces unnecessary computation, and generates meaningful review comments that promote best practices and independent problem-solving. The approach addresses key limitations of traditional automated evaluation methods by providing explanation-oriented feedback rather than simple test-case results.

To further enhance code understanding, an NLP-based automatic comment generation module was introduced as an extension, enabling the system to produce descriptive inline comments for undocumented code. Experimental evaluation using BERT-Score achieved an F1-score of 88%, demonstrating strong semantic alignment with expert feedback. The implementation using a Flask-based web interface confirms the practicality and scalability of the system in real-world educational environments. Overall, the proposed framework offers an effective and reliable solution for improving programming education through automated, learning-oriented code feedback.

6. FUTURE SCOPE

The proposed system can be further enhanced by extending support to a wider range of programming languages and integrating deeper semantic analysis for complex code structures. Future work may include incorporating real-time interactive tutoring capabilities, where the system provides step-by-step guidance based on learner queries and coding progress. Additionally, fine-tuning the GPT model with domain-specific educational datasets can

improve feedback accuracy and personalization for different learner levels.

Further improvements can focus on integrating advanced evaluation metrics beyond BERT-Score, such as human-in-the-loop assessment and adaptive learning analytics to measure learning outcomes more effectively. The NLP-based comment generation module can also be enhanced to generate more detailed documentation, including function summaries and code explanations in multiple languages. Finally, deploying the system as a scalable cloud-based educational platform can enable widespread adoption in classrooms, online learning systems, and competitive programming environments.

REFERENCES

1) M. of Science and N. I.S. A. ICT. (2021). Global Digital Competency Education. Accessed: Dec. 2023.
2) OECD Skills Outlook 2019: Thriving in a Digital World, O. for Economic Co-operation and Development, OECD, Paris, France, May 2019.
3) M. Cardona, R. Rodriguez, and K. Ishmael. (May 2025). Artificial Intelligence and the Future of Teaching and Learning: Insights and Recommendations. Office of Educational Technology, U.S. Department of Education. Accessed: May 2025.
4) U.S. Department of Education. (2025). Artificial Intelligence (AI) Guidance and Use Case Inventory. Accessed: May 3, 2025.
5) Ministry of Education and Culture Helsinki. (2023). Policies for the Digitalisation of Education and Training Until 2027. Accessed: May 2025.
6) Ministry of Education Singapore. (2024). Transforming Education Through Technology: Masterplan 2030. Accessed: May 3, 2025.
7) M. of Education. (2024). Exploring the 2022 Revised Elementary School Curriculum Organization and Operation. Accessed: Dec. 2023.
8) British Informatics Olympiad. (2025). Rules of the 2025 British Informatics Olympiad. Accessed: May 2025. .
9) College Board. (2023). Ap Computer Science a—The Exam. AP Central. Accessed: May 2025.
10) S. Choi, D. Lee, J. Kim, Y. Jang, and H. Kim, ‘‘Designing llm-based code reviewing learning environment for programming education,’’ J. Korean Assoc. Comput. Educ., vol. 26, no. 5, pp. 1–11, Jul. 2023.
11) M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman, ‘‘CodeAid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs,’’ in Proc. CHI Conf. Hum. Factors Comput. Syst. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 1–20, doi: 10.1145/3613904.3642773.
12) S. Sahai, U. Z. Ahmed, and B. Leong, ‘‘Improving the coverage of GPT for automated feedback on high school programming assignments,’’ in Proc. NeurIPS Workshop Generative AI Educ.

(GAIED), vol. 46. Baton Rouge, LA, USA: MIT Press, 2023.

13) M.-F. Wong, S. Guo, C.-N. Hang, S.-W. Ho, and C.-W. Tan, ‘‘Natural language generation and understanding of big code for AI-assisted programming: A review,’’ Entropy, vol. 25, no. 6, p. 888, Jun. 2023, doi: 10.3390/e25060888.

14) H. Nguyen and V. Allan, ‘‘Using GPT-4 to provide tiered, formative code feedback,’’ in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ., Mar. 2024, pp. 958–964.

15) S. Saylam, N. Duman, Y. Yildirim, and K. Satsevich, ‘‘Empowering education with AI: Addressing ethical concerns,’’ London J. Social Sci., vol. 6, no. 6, pp. 39–48, Sep. 2023.

16) M. Jaber, ‘‘New innovations in higher education’s academic integrity and classroom strategies,’’ in Academic Integrity in the Age of Artificial Intelligence. Hershey, PA, USA: IGI Global, Feb. 2024, pp. 281–300.

17) Y. Xie, S. Wu, and S. Chakravarty, ‘‘AI meets AI: Artificial intelligence and academic integrity—A survey on mitigating AI-assisted cheating in computing education,’’ in Proc. 24th Annu. Conf. Inf. Technol. Educ., Oct. 2023, pp. 79–83.

18) B. McDanel and E. Novak, ‘‘Designing LLM-resistant programming assignments: Insights and strategies for CS educators,’’ in Proc. 56th ACM Tech. Symp. Comput. Sci. Educ. New York, NY, USA: Association for Computing Machinery, Feb. 2025, pp. 756–762, doi: 10.1145/3641554.3701872.

19) J. P. Amos, O. A. Amodu, R. A. R. Mahmood, A. B. Abdulqudus, A. F. Zakaria, A. R. Iyanda, U. A. Bukar, and Z. M. Hanapi, ‘‘A bibliometric exposition and review on leveraging LLMs for programming education,’’ IEEE Access, vol. 13, pp. 58364–58393, 2025.

20) W. Dai, J. Lin, H. Jin, T. Li, Y.-S. Tsai, D. Gasevic, and G. Chen, ‘‘Can large language models provide feedback to students? A case study on ChatGPT,’’ in Proc. IEEE Int. Conf. Adv. Learn. Technol. (ICALT), Jul. 2023, pp. 323–325.

21) R. Singh, S. Gulwani, and A. Solar-Lezama, ‘‘Automated feedback generation for introductory programming assignments,’’ in Proc. 34th ACM SIGPLAN Conf. Program. Lang. Design Implement. New York, NY, USA: Association for Computing Machinery, Jun. 2013, pp. 15–26, doi: 10.1145/2491956.2462195.

22) J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, ‘‘LLaMA-reviewer: Advancing code review automation with large language models through parameterefficient fine-tuning,’’ 2023, arXiv:2308.11148.

23) Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, ‘‘A critical review of large language model on software engineering: An example from ChatGPT and automated program repair,’’ 2023, arXiv:2310.08879.

24) T. Sun, J. He, X. Qiu, and X. Huang, ‘‘BERTScore is unfair: On social bias in language model-based metrics for text generation,’’ in Proc. Conf. Empirical

Methods Natural Lang. Process., vol. 1, Jan. 2022, pp. 3726–3739s.

25) T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, ‘‘BERTScore: Evaluating text generation with BERT,’’ 2019, arXiv:1904.09675.

26) S. Crandall, G. Sprint, and B. Fischer, ‘‘Generative pre-trained transformer (GPT) models as a code review feedback tool in computer science programs,’’ J. Comput. Sci. Colleges, vol. 39, no. 1, pp. 38–47, Oct. 2023.

27) S. Crandall, B. Fischer, and J. P. Crandall, ‘‘WIP: ARTful insights from a pilot study on GPT-based automatic code reviews in undergraduate computer science programs,’’ in Proc. IEEE Frontiers Educ. Conf. (FIE), Oct. 2024, pp. 1–5.

28) C3Coding. (2023). The Online Judge System for Elementary and Secondary School Students. Accessed: Dec. 2023.

29) H.-J. Moon, ‘‘Evaluation of user satisfaction and effectiveness of web (App)-based artificial intelligence application services,’’ J. Digit. Contents Soc., vol. 24, no. 10, pp. 2611–2617, Oct. 2023.

30) Babburi, S. (2024). Explainable AI Framework for Policy-Compliant Anomaly Detection in Data Pipelines.

31) Todupunuri, A. (2025). IMPROVING CUSTOMER EXPERIENCE WITH MODERN BANKING SOLUTIONS. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5120615

32) Gaddam, S. From Fixed Specifications to Self-Adapting Systems: A Machine Learning Perspective on Software Engineering.

33) Vasagam, M. (2024, August 30). Ensuring security in modern data pipelines: Practical strategies for data engineers. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2401.

34) Mudusu, S. K., & Gentyala, S. (2026). Zero-Trust Data Pipelines for AI Systems: A Framework for Secure, Verifiable, and Auditable Data Engineering. JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE), 14(2), 10-25.

35) Santthosh Saai Reddy Purmani. (2026). Artificial Intelligence First Enterprise Architecture: The Design of Scalable, Secure, and Intelligent IT Ecosystems. American Journal of AI Cyber Computing Management, 6(1(2)), 1–8. https://doi.org/10.64751/ajaccm.2026.v6.n1(2).pp1-8

36) Purmani, S. S. R. (2025). Streamlining IT operations and service management with agile frameworks. European Journal of Advances in Engineering and Technology, 12(4), 76–81.

37) Mudusu, S. K. (2025). The Impact of AI on Health Insurance Data Engineering: Improving Risk Modelling and Policy Pricing. JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE), 13(1), 99-107.

38) Purmani, S. S. R. (2025). Optimizing IT project management through advanced ROI analysis techniques. International Journal for

Innovative Engineering and Management Research, 14(3), 301–312.

39) Kotte, G. (2025). Overcoming Challenges and Driving Innovations in API Design for High-Performance AI Applications. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283649

40) Mudusu, S. K. (2026, March 26). A data trust scoring framework for reliable and responsible AI systems. InfoWorld (Foundry Expert Contributor Network).

41) Purmani, S. S. R. (2024). Aligning IT investment decisions with overall business strategy from an enterprise program management perspective, focusing on the integration of IT leadership in strategic decision-making processes. International Journal of Communication Networks and Information Security, 16(5), 1213–1219

42) Mudusu, S. K. (2024, August). Achieving fully autonomous AI-driven data pipelines: Exploring zero-touch automation for efficient and scalable data engineering solutions. International Journal on Recent and Innovation Trends in Computing and Communication, 12(2), 1182–1186.

43) Mahtabi, M., Roshan, M., Muhit, M. M. I., Behvar, A., & Haghshenas, M. (2026). Cryogenic ultrasonic fatigue: Mechanisms, advancements, and insights. Cryogenics, 153, 104257. https://doi.org/10.1016/j.cryogenics.2025.104257

44) Purmani, S. S. R. (2025). Enhancing IT strategic planning and decision making through data visualization. International Journal of Enhanced Research in Management & Computer Applications, 14(4), 75–81

45) Mudusu, S. (2025). Health Insurance Fraud Detection: The Role Of Advanced It Systems In Preventing And Identifying Fraud. International Journal, 16(1), 3769-3777

46) GIRISH KOTTE. (2025). ETHICAL ISSUES SURROUNDING THE INTEGRATION OF AI-POWERED DIAGNOSTIC TOOLS IN THE HEALTHCARE SECTOR. American Journal of AI Cyber Computing Management, 5(4), 329–334. https://doi.org/10.64751/ajaccm.2025.v5.n4.pp329-334

47) Mudusu, S. K. (2023, July 19). Context-aware cognitive data fabrics: Enhancing AI pipeline orchestration for real-time inference. International Journal of Communication Networks and Information Security, 15(4), 738–745.

48) Mahimalur, R. K., Vasgam, M., & Manoharan, D. Devops Lifecycle Management And Cloud Migration Assessments: A Security-Driven CICD Perspective.

49) DEVARASETTY, N. (2023). SCALABLE DATA ENGINEERING APPROACHES FOR AI-DRIVEN INDUSTRIAL IOT APPLICATIONS. INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH AND MANAGEMENT, 11(06), 954-968.

50) Manoharan, D. (2026). AI-Driven Anomaly Detection Models for Preventing Claims Denials and Revenue Leakage in Healthcare. Available at SSRN 6385759.

51) Mudusu, S. K. (2025). AI-driven data engineering in the Internet of Things: Scaling data pipelines for smart device ecosystems. ISCSITR-International Journal of Data Engineering (ISCSITR-IJDE), 6(1), 1–9.

52) Purmani, S. S. R. (2024). Aligning IT investment decisions with overall business strategy from an enterprise program management perspective, focusing on the integration of IT leadership in strategic decision-making processes. International Journal of Communication Networks and Information Security, 16(5), 1213–1219

53) Kotte, G. (2025). Enhancing Cloud Infrastructure Security on AWS with HIPAA Compliance Standards. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283660

54) Maturi, S. Y. (2025). Vulnerabilities in the 802.11 Wireless Client Selection Mechanis.

55) Agrawal, A. M., Gajula, S., Shinde, R. P., Shah, H., & Ghosh, H. (2025, July). Machine Translation for Long Sequences with Enhanced Attention Mechanisms. In 2025 5th International Conference on Electrical, Computer and Energy Technologies (ICECET) (pp. 1-6). IEEE.

56) Subramanian, V. K., Bhambri, S., & Gajula, S. (2025, April). Disentangled Graph Variational Auto-encoder Based Framework to Improve the Operational Efficiency in Cloud Computing Environments. In International Conference on Computer Vision and Robotics (pp. 396-407). Cham: Springer Nature Switzerland.

57) Mudusu, S. K. Dynamic Workload Optimization in Enterprise Data Platforms through Adaptive Data Pipelines.

58) Mahtabi, M., Roshan, M., Muhit, M. M. I., Behvar, A., & Haghshenas, M. (2026). Cryogenic ultrasonic fatigue: Mechanisms, advancements, and insights. Cryogenics, 153, 104257. https://doi.org/10.1016/j.cryogenics.2025.104257

59) Mudusu, S. K. (2026, April 15). The secure intelligence framework: Architecting AI systems for a data-driven world. CIO (Foundry Expert Contributor Network).

60) Maturi, S. Y. (2023). Crowdsourced frontier: Unveiling autonomous adversarial cybercapabilities via open AI competition. International Journal of Intelligent Systems and Applications in Engineering, 11(1s), 275–284.

61) Sikder, M. Z., Shakil, M. A. I., Ahad, A., Karim, M. F., Intakhab, B., & Islam, D. A. (2025, June). Microwave-Based Detection of Early-Stage Renal Cell Carcinoma Using UHF Range Antenna. In 2025 International Conference on Computer Systems and Technologies (CompSysTech) (pp. 1-6). IEEE.

62) Ranjbareslamloo, S., Dzukey, G. A., Islam Muhit, M. M., & Qattawi, A. (2025). Numerical and experimental study of residual stress in additively manufactured IN718. Manufacturing Letters, 44, 915–927. https://doi.org/10.1016/j.mfglet.2025.06.108

63) Mudusu, S. K. (2022, September). Ensuring data reliability in AI systems: Connecting data quality and model integrity.

International Journal for Innovative Engineering and Management Research, 11(9), 318–325.

64) Manoharan, D. (2025). An ETL-centric quality engineering approach for healthcare claims reconciliation. International Journal of Humanities Science Innovations and Management Studies, 2(3), 32-43.

65) Mudusu, S. K. (2025). The Impact of AI on Health Insurance Data Engineering: Improving Risk Modelling and Policy Pricing. JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE), 13(1), 99-107.