International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 5, May 2026)

Code Sage-AI: A Context-Aware Generative Framework for Automated Code Review and Intelligent Comment Synthesis

[1]V. Narendra Chary, [2]Dr. Sd.Gulam Gouse

M.Tech Scholar, Department of CSE, Nimra College of Engineering and Technology, Jupudi, Ibrahimpatnam, Vijayawada, India
Professor, Department of CSE, Nimra College of Engineering and Technology, Jupudi, Ibrahimpatnam, Vijayawada, India

Abstract- Code review is a critical process in software development for ensuring code quality, readability, and maintainability. However, traditional manual review is time-consuming, inconsistent, and difficult to scale with growing code complexity. Existing automated tools mainly rely on rule-based or static analysis methods, which lack the ability to understand code semantics and generate meaningful feedback.

This paper proposes a transformer-based automated code review system using a fine-tuned Generative Pre-trained Transformer (GPT) model. The system analyzes source code and generates context-aware, human-like review comments. In addition, an NLP-based module is incorporated to produce line-by-line explanatory comments, improving code understanding. A Flask-based web interface enables real-time interaction and usability.

The model is evaluated using BERTScore and achieves an F1-score of 0.8807, indicating high semantic accuracy and relevance. The proposed system reduces manual effort, improves consistency, and provides a scalable solution for automated code review in modern software development environments.

Keywords - Automated Code Review, Generative Pre-trained Transformer (GPT), Transformer Models, Natural Language Processing (NLP), Code Comment Generation, Software Quality, BERTScore, Deep Learning, Code Analysis, Flask Web Application.

I. INTRODUCTION

Code review is a fundamental activity in the software development lifecycle that ensures code quality, correctness, and maintainability. It helps identify defects, enforce coding standards, and improve overall software reliability. Traditionally, code review is performed manually by developers, which makes the process time-consuming, inconsistent, and highly dependent on individual expertise. As software systems grow in size and complexity, manual review becomes difficult to scale and may lead to overlooked issues or delayed development cycles.

To address these challenges, automated code review tools such as static analyzers and linters have been introduced. These tools can efficiently detect syntax errors and rule violations; however, they are limited in their ability to understand the context and intent of the code. As a result, they cannot generate meaningful suggestions or human-like explanations, which are essential for improving code readability and developer understanding.

Recent advancements in artificial intelligence, particularly in Natural Language Processing (NLP), have enabled the development of models capable of understanding and generating human-like text. Transformer-based architectures, such as Generative Pre-trained Transformer (GPT) models, have shown remarkable performance in capturing contextual relationships within sequences.

These models are widely used in tasks such as text generation, summarization, and translation, and are increasingly being applied to source code analysis.

In this work, a transformer-based automated code review system is proposed using a fine-tuned GPT model. The system is designed to analyze source code and generate context-aware review comments that resemble human feedback. In addition, an NLP-based module is incorporated to provide line-by-line explanatory comments, enhancing code readability and understanding. A web-based interface developed using Flask enables users to upload code and receive real-time feedback.

The main contributions of this work include:

- Development of a GPT-based system for automated code review
- Generation of context-aware and human-like review comments
- Integration of an NLP module for detailed code explanation
- Implementation of a user-friendly web interface for practical use
- Evaluation using semantic similarity metrics to ensure output quality

Overall, the proposed approach aims to reduce manual effort, improve consistency in code review, and provide a scalable solution for enhancing software quality in modern development environments.

II. LITERATURE REVIEW

Automated code review has gained significant attention as a means to improve software quality and reduce manual effort. Early approaches primarily relied on rule-based systems and static analysis tools, which detect syntax errors, code smells, and violations of predefined coding standards. While these methods are efficient for basic validation, they lack the ability to understand the semantics and intent of the code, limiting their effectiveness in providing meaningful feedback.

With the advancement of machine learning, various models such as Support Vector Machines (SVM), Decision Trees, and Random Forest have been applied to tasks like defect prediction and code classification. These approaches improve accuracy compared to traditional methods by learning patterns from data. However, they depend heavily on feature engineering and are limited in capturing deeper contextual relationships within code.

Deep learning techniques have further enhanced code analysis capabilities. Models such as Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks have been used to process sequential data and understand code structure. Although these models improve performance, they struggle with long-range dependencies and large-scale codebases, which restricts their applicability.

The introduction of transformer architectures has significantly advanced the field of code understanding. Transformer-based models, including GPT, CodeBERT, and CodeT5, utilize self-attention mechanisms to capture contextual relationships more effectively than previous models. These models have demonstrated strong performance in tasks such as code summarization, defect detection, and automated review generation by producing human-like textual outputs.

Recent research highlights the effectiveness of GPT-based models in generating context-aware review comments that closely resemble human feedback.

Evaluation metrics such as BERTScore have been used to measure semantic similarity between generated and reference comments, providing a more reliable assessment compared to traditional lexical metrics.

Despite these advancements, many existing systems focus primarily on high-level review generation and do not provide detailed, line-by-line explanations. Additionally, practical deployment and integration into real-world development workflows remain limited.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 5, May 2026)

To address these gaps, the proposed system integrates a fine-tuned GPT model for automated code review with an NLP-based module for line-level comment generation. This combined approach enhances both code quality and readability while providing a scalable and practical solution for modern software development environments.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Code review is an essential step in the software development lifecycle for ensuring code quality, correctness, and maintainability. However, traditional code review is predominantly manual, making it time-consuming, inconsistent, and dependent on the expertise of individual developers. As modern software systems grow in size and complexity, manual review processes become difficult to scale and often lead to overlooked defects or delayed development cycles.

Existing automated tools, such as static analyzers and linters, provide partial solutions by detecting syntax errors and rule violations. While these tools are efficient, they lack the ability to understand the context and intent of the code. As a result, they cannot generate meaningful, human-like feedback or provide explanations that help developers improve code quality and readability.

Furthermore, many current automated approaches focus only on high-level analysis and do not offer detailed line-by-line explanations of code. This limits their usefulness, especially for beginner developers who require additional guidance to understand code behavior. Additionally, the lack of integrated systems that combine semantic understanding with user-friendly interfaces restricts real-world adoption.

Therefore, there is a need for an intelligent and scalable solution that can:

- Understand code semantics and context
- Generate meaningful, human-like review comments
- Provide detailed line-by-line explanations
- Support real-time interaction through a user-friendly interface
- Improve overall code quality and developer productivity

B. Objectives

The primary objective of this research is to develop an intelligent automated code review system using transformer-based deep learning techniques. The specific objectives are as follows:

- To design and implement a GPT-based model for automated code review
- To generate context-aware and human-like review comments for source code
- To develop an NLP-based module for line-by-line comment generation
- To enhance code readability and understanding through automated explanations
- To integrate the system with a Flask-based web interface for real-time usability
- To evaluate the system using semantic similarity metrics such as BERTScore
- To improve consistency and reduce manual effort in code review processes

IV. PROPOSED METHODOLOGY

A. System Overview

The proposed system is designed to automatically analyze source code and generate review comments using a fine-tuned GPT-based transformer model. The system integrates a web-based interface with backend processing modules to provide real-time code review and comment generation.

The workflow consists of the following steps:

1. User uploads source code file
2. Code is preprocessed and converted into input format
3. GPT model generates review comments

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 5, May 2026)

4. NLP module generates line-by-line explanations
5. Results are displayed to the user

B. Input Representation

Let the input source code be represented as:

$$C = \{c_1, c_2, c_3, \ldots, c_n\}$$

where each $c_i$represents a line or token of the source code.

The input is transformed into a structured prompt:

$$P = C + \text{“<|endoftext|> Comment:”}$$

This prompt is passed to the GPT model for generating review comments.

C. Data Preprocessing

To ensure proper model performance, preprocessing steps are applied:

- Removal of unwanted characters
- Tokenization using GPT tokenizer
- Padding and truncation of sequences
- Conversion into numerical token IDs

The tokenized input is represented as:

$$T = \{t_1, t_2, t_3, \ldots, t_m\}$$

D. Model Training and Generation

A transformer-based causal language model is used for generating review comments. The model is fine-tuned using supervised learning on a code review dataset.

The objective is to maximize the probability of generating the correct review comment Rgiven input C:

$$P(R \mid C) = \prod_{i=1}^{k} P(r_i \mid C, r_1, r_2, \ldots, r_{i-1})$$

where $r_i$represents each token in the generated review.

During inference, the model generates output using sampling techniques such as:

- Top-k sampling
- Temperature scaling

E. NLP-Based Comment Generation

In addition to GPT-based review generation, an NLP module is used to produce line-level comments.

The process includes:

- Identifying assignment statements
- Detecting variable names and values
- Classifying data types (integer, float, string)
- Applying tokenization and part-of-speech tagging

Example rule:

- If a line contains assignment → generate comment:

  “This line assigns a value to a variable”

This enhances code readability and understanding.

F. System Implementation

The system is implemented using:

- **Backend:** Python
- **Framework:** Flask
- **Libraries:** Transformers, NLTK, Pandas
- **Model Loading:** AutoTokenizer, AutoModelForCausalLM

The uploaded file is temporarily stored and processed for analysis. The generated output is displayed in a structured format including:

- Input code
- GPT-generated review
- NLP-generated comments

G. Performance Evaluation

The system is evaluated using **BERTScore**, which measures semantic similarity between generated and reference comments.

$$\text{BERTScore} = \frac{2 \cdot P \cdot R}{P + R}$$

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 5, May 2026)

where:

- *P*= Precision
- *R*= Recall

This metric ensures that the generated comments are contextually accurate and meaningful.

V. RESULTS AND DISCUSSION

A. Experimental Setup

The proposed system is implemented using **Python** with a Flask-based web framework for user interaction. The transformer model is developed using the **Hugging Face Transformers library**, and the NLP module is implemented using **NLTK** for text processing.

The dataset used for training consists of code snippets and corresponding review comments. The model is trained on a subset of approximately **3000 samples** to learn the relationship between source code and human-written feedback.

B. Performance Evaluation

The performance of the system is evaluated using **BERTScore**, which measures semantic similarity between generated comments and reference comments. This metric is preferred over traditional methods as it captures contextual meaning.

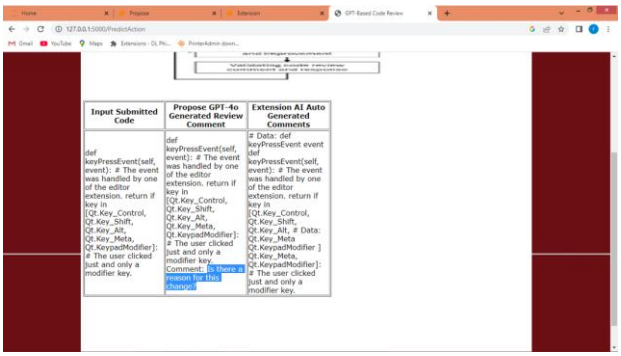
Table 1: Performance Metrics

Metric	Value
Precision	0.88
Recall	0.88
F1-Score	**0.8807**

The results indicate that the model produces review comments that are highly similar to human-generated feedback in terms of meaning and context.

C. Output Analysis

The system generates two types of outputs:



1. **GPT-Based Review Comments**
   - Provides high-level feedback on code quality
   - Suggests improvements and identifies issues
2. **NLP-Based Inline Comments**
   - Explains individual lines of code
   - Identifies variables and operations

Table 2: Sample Output

Input Code	Generated Review	AI Comments
Python snippet	Suggests optimization	Line-level explanation

The combined output improves both **code quality** and **code understanding**.

D. Discussion

The experimental results demonstrate that the proposed system is effective in generating meaningful and context-

aware review comments. The GPT-based model captures semantic relationships within the code and produces human-like feedback.

The integration of the NLP module further enhances the system by providing detailed explanations at the line level. This makes the system particularly useful for beginners and developers who require additional clarity.

However, the system has certain limitations. It performs best on small to medium-sized code snippets and may require optimization for large-scale codebases. Additionally, the NLP module uses rule-based logic, which may not fully capture complex programming constructs.

VI. CONCLUSION

This paper presented a transformer-based automated code review system using a fine-tuned Generative Pre-trained Transformer (GPT) model. The proposed framework addresses the limitations of traditional manual and rule-based approaches by enabling semantic understanding of source code and generating context-aware, human-like review comments.

The integration of an NLP-based module further enhances the system by providing line-by-line explanatory comments, improving code readability and developer understanding. The use of a Flask-based web interface ensures practical usability, allowing users to upload code and receive real-time feedback.

Experimental results, evaluated using BERTScore, demonstrate that the system achieves high semantic similarity with human-written reviews, indicating strong contextual accuracy and relevance. The proposed approach effectively reduces manual effort, improves consistency in code review, and enhances overall software quality.

However, the system performs best on small to medium-sized code snippets and requires further optimization for large-scale codebases and complex applications. Overall, the framework provides a scalable and efficient solution for automated code review and can be extended for real-world deployment in modern software development workflows.

VIII. FUTURE SCOPE

The proposed automated code review system can be further enhanced by integrating it with version control platforms such as GitHub and GitLab, enabling automatic analysis of pull requests and continuous monitoring of code quality during development. This would support seamless adoption in real-world software engineering workflows.

Future work can also focus on extending the system to support multiple programming languages and handling large-scale codebases more efficiently. Incorporating more advanced transformer architectures or larger pre-trained models can further improve the quality and depth of generated review comments.

The NLP-based comment generation module can be improved by adopting deep learning techniques to better understand complex programming constructs and generate more context-aware explanations. Additionally, integrating static analysis and security vulnerability detection can enhance the system's capability to identify potential risks in code.

Another important direction is deploying the system within continuous integration and continuous deployment (CI/CD) pipelines, allowing real-time code evaluation during the development lifecycle. The inclusion of dashboards and visualization tools can help developers track code quality metrics and improvements over time.

Overall, these enhancements can make the system more robust, scalable, and suitable for practical deployment in modern software development environments.

REFERENCES

[1] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[2] T. Brown *et al.*, "Language models are few-shot learners," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 1877–1901.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–37, 2018.

[4] Z. Feng *et al.*, "CodeBERT: A pre-trained model for programming and natural languages," in *Proc. Findings of EMNLP*, 2020, pp. 1536–1547.

[5] Y. Liu *et al.*, "RoBERTa: A robustly optimized BERT pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[6] T. Zhang *et al.*, "BERTScore: Evaluating text generation with BERT," in *Proc. International Conference on Learning Representations (ICLR)*, 2020.

[7] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proc. International Conference on Software Engineering (ICSE)*, 2012, pp. 837–847.

[8] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., Pearson, 2023.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD*, 2016, pp. 785–794.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.