IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

REAL-TIME CODE DEBUGGING, SUGGESTIONS AND LEARNING ASSISTANT USING AI

DR.G.D.K.Kishore[1],K.Mahitha[2],I.Akhheela [2],K.Leela Venkata Sai Krishna[2],V.Durga Prasad[2]

Associate Professor & HOD[1], Department of Information Technology,SRKInstituteofTechnology,NTR,AndhraPradesh, India

Student[2], Department of Information Technology,SRKInstituteofTechnology,NTR,AndhraPradesh, India

ABSTRACT

The rapid advancement of software development technologies has significantly increased the complexity and scale of modern codebases. As a result, debugging and code comprehension have become more time-consuming and error-prone, particularly for novice programmers. This paper presents a novel AI-powered system designed to provide real-time code debugging, intelligent suggestions, and personalized learning support. Leveraging machine learning models, natural language processing, and static/dynamic code analysis techniques, the proposed assistant can detect common coding errors, offer context-aware recommendations, and explain programming concepts interactively. The system is integrated into popular development environments, enabling seamless and intuitive user interaction. Through extensive testing across various programming languages and real-world scenarios, our tool demonstrates improved debugging efficiency, faster learning curves, and reduced dependency on external documentation. This research contributes to the evolution of intelligent programming assistants, aiming to bridge the gap between coding education and professional development practices.

Keywords: Real-time debugging, code suggestions, AI assistant, programming education, intelligent IDE, machine learning, code analysis, software development tools, NLP in programming, developer productivity.

INTRODUCTION

Writing and debugging code can be challenging, especially for beginners who struggle with understanding errors and finding efficient solutions. Traditional tools often lack real-time support and personalized guidance. With the rise of artificial intelligence, particularly machine learning and natural language processing, it's now possible to build smart assistants that can help developers as they code.

This paper presents an AI-powered assistant that provides real-time code debugging, intelligent suggestions, and learning support. Integrated into common development environments, the system detects errors, suggests fixes, and explains coding concepts in a user-friendly manner. The goal is to improve productivity, reduce debugging time, and enhance the learning experience for developers at all levels.

PROBLEM STATEMENT

Despite the availability of modern development tools, developers—particularly beginners—continue to face significant challenges in identifying and resolving code errors efficiently. Traditional debugging methods often require manual effort, repeated trial-and-error, and consultation of external resources, leading to increased development time and frustration. Additionally, existing code editors and IDEs offer limited real-time guidance and lack adaptive learning support tailored to individual skill levels.

There is a need for an intelligent system that can provide real-time debugging, suggest meaningful improvements, and offer contextual learning support as code is being written. Such a solution should not only reduce the time spent on debugging but also enhance code quality and facilitate continuous learning within the development environment.

MOTIVATION

The system leverages a combination of **machine learning**, **natural language processing (NLP)**, and **code analysis techniques** to understand, analyze, and improve source code dynamically. It monitors the code in real-time and provides helpful suggestions and explanations without interrupting the development flow.

Key Features:

1. **Real-Time Error Detection and Debugging**
   Instantly identifies syntax and logical errors as the user writes code, with precise highlighting and descriptive messages.
2. **AI-Based Code Suggestions**
   Offers intelligent code completions, optimizations, and refactoring suggestions based on learned patterns from large codebases.
3. **Context-Aware Explanations**
   Provides simple, natural-language explanations of errors and programming concepts to support learning.
4. **Integrated Learning Support**
   Recommends tutorials, documentation, or example snippets relevant to the user's code and error context.
5. **Language and IDE Compatibility**
   Supports multiple programming languages and integrates with popular IDEs (e.g., VS Code, PyCharm) for a smooth development experience.

LITERATUREREVIEW

1. Deep Learning-based Error Classification

- **Author**: Gupta et al. (2021)
- **Title**: Deep Learning-based Error Classification
- **Description**:
  This study explores the use of **deep learning architectures** like **Convolutional Neural Networks (CNN)** and **Long Short-Term Memory (LSTM)** networks to automatically classify coding errors. The model is trained using the **CodeXGLUE** benchmark dataset, which contains code snippets with associated labels for various error types. CNN helps extract local code features, while LSTM captures temporal patterns and sequence-based context in the code. This combination allows the system to learn common error patterns and classify them into predefined categories such as syntax, type, or structure-related errors.
  The approach demonstrates how deep learning can be applied to static code analysis, focusing mainly on textual patterns in source code.

- **Limitations**:
  The model performs well when handling **syntax-level issues**, but it is **limited in scope** when it comes to detecting **semantic errors** or logic bugs. Since it doesn't understand deeper code semantics or intent, it struggles to identify issues that are not explicitly reflected in the structure or syntax.

2. Transformer-based Bug Localization

- **Author**: Wang et al. (2022)
- **Title**: Transformer-based Bug Localization
- **Description**:
  This paper utilizes powerful **transformer-based models**, specifically **CodeBERT** and **GraphCodeBERT**, for the task of **bug localization**. These models are pretrained on large-scale code corpora and then fine-tuned using the **Defects4J** dataset, which consists of real bugs collected from Java projects.
  CodeBERT captures the syntactic and semantic relationships in code, while GraphCodeBERT adds structural insights using the code's **abstract syntax tree (AST)** or control flow graph. The system analyzes a given codebase and pinpoints buggy lines or segments based on learned representations, offering high accuracy and generalization across different projects.
- **Limitations**:
  One significant challenge is that the model **struggles with unseen or rare error types**. Since transformers rely heavily on patterns observed during training, their performance drops when dealing with bugs that are structurally or semantically different from the training examples.

3. Reinforcement Learning for Debugging

- **Author**: Li et al. (2020)
- **Title**: Reinforcement Learning for Debugging
- **Description**:
  This research introduces a **Reinforcement Learning (RL)** framework for automated debugging, where agents learn to fix bugs by interacting with the code environment and receiving rewards for correct actions. Algorithms such as **Q-Learning** and **Deep Q-Networks (DQN)** are used to learn optimal debugging strategies over time.
  The model is trained on the **ManyBugs** dataset, which contains multiple versions of buggy and fixed C programs. The RL agent navigates through the code, detects anomalies, and suggests corrective actions that align with fixes observed in the dataset. This technique mimics human-like debugging, with the added advantage of self-improvement through exploration and feedback.
- **Limitations**:
  A key downside of RL-based systems is the **high computational cost**. Training RL agents typically requires significant computing resources and time, especially when exploring large codebases or complex reward structures. This can make the approach less feasible for real-time applications or integration into lightweight developer tools.

4. Graph Neural Networks for Code Analysis

- **Author**: Zhou et al. (2023)
- **Title**: Graph Neural Networks for Code Analysis
- **Description**:
  This work proposes the application of **Graph Neural Networks (GNNs)** for deeper structural analysis of source code. Code is transformed into **graph representations**, such as Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), or Data Flow Graphs (DFGs), which GNNs process to learn node-level and graph-level embeddings.
  The model uses **AST-based architectures** to understand the relationships between different code elements. It is trained on **GitHub Issues** datasets that link actual bug reports with corresponding code changes. The result is a system capable of contextual code understanding and bug detection across multiple layers of abstraction.
- **Limitations**:
  GNNs require **extensive labeled datasets** to perform well. Obtaining high-quality, labeled code data (e.g., correctly mapped bug reports to code changes) at scale is a time-consuming and labor-intensive process. Without such data, the model's performance and generalization capability are limited.

5. Hybrid AI-based Debugging (Proposed Model)

- **Author**: Our Proposed Model
- **Title**: Hybrid AI-based Debugging using GNN + Transformers + Reinforcement Learning
- **Description**:
  This proposed model integrates the strengths of **multiple AI paradigms**:
  - **CodeBERT** (for semantic understanding of code),
  - **GNNs** (for structural analysis),
  - **Reinforcement Learning with Human Feedback (RLHF)** (to guide code fixing via rewards),
  - **Bayesian Inference** (to deal with uncertainty in predictions and improve decision-making).
    By leveraging both **CodeXGLUE** and **Defects4J** datasets, the system performs holistic debugging—starting from identifying and localizing bugs to recommending accurate fixes. This ensemble approach helps improve both precision and coverage, providing a robust and scalable solution for real-time code debugging environments.
- **Limitations**:
  Despite its **high accuracy (92.4%)**, the hybrid model presents challenges in terms of **explainability**—making it hard for developers to trust or understand some AI decisions. It also faces **adaptability issues** when applied in real-world software projects, where codebases are diverse, domain-specific, and less structured than in benchmark datasets.

EXISTINGSYSTEM:

Several code editors and integrated development environments (IDEs) offer basic support for syntax highlighting, code completion, and static error detection. Tools like **Visual Studio Code**, **Eclipse**, and **PyCharm** include built-in debugging features and plugins that assist developers during coding. Additionally, platforms such as **Stack Overflow**, **GitHub Copilot**, and **Linting tools** (e.g., ESLint, Pylint) provide external support in the form of code suggestions, error detection, and community-driven help.

- **Lack of Real-Time Adaptive Learning**: Most tools do not adapt their feedback based on the user's expertise level or learning curve.
- **Limited Conceptual Explanation**: Existing systems rarely explain the logic or reason behind a suggestion or error, leaving beginners confused.
- **Dependence on Internet Resources**: Many tools require users to search for help externally, interrupting the coding flow.
- **No Unified Learning + Debugging Integration**: There is often a clear separation between debugging tools and educational resources, which can slow down learning.

PROPOSEDSYSTEM:

The proposed system is an AI-driven assistant designed to provide developers with real-time support during code development. It integrates advanced machine learning, natural language processing, and static/dynamic code analysis techniques to deliver intelligent debugging, code suggestions, and personalized learning assistance within the development environment.

Unlike traditional tools, this system continuously analyzes the code as it is being written, instantly identifying errors, suggesting corrections, and explaining the underlying issues in simple, understandable language. It also adapts to the user’s level of expertise, offering detailed guidance for beginners and concise suggestions for experienced developers.

METHODOLOGY:

The development of the proposed AI-based real-time code debugging and learning assistant follows a structured, modular approach. The system architecture is divided into several components that work together to analyze code, generate suggestions, and support learning in real-time. The methodology consists of the following key stages:

1. Data Collection and Preprocessing

- Large datasets of code snippets, error logs, and programming queries are collected from open-source platforms such as GitHub, Stack Overflow, and public coding forums.
- The data is cleaned, labeled, and structured for supervised and unsupervised learning tasks.

2. Model Training

- **Machine Learning Models** are trained to detect common coding errors and recommend corrections.
- **Natural Language Processing (NLP)** models are fine-tuned (e.g., using transformer-based architectures like BERT or GPT) to understand and generate natural language explanations of errors and suggestions.
- A classification model is trained to identify user skill levels based on coding patterns and interaction history.

3. Real-Time Code Analysis

- As the user writes code in an IDE, the system performs live code parsing and analysis.
- A static analysis engine detects syntax and logical errors, while a dynamic component tracks runtime behavior in supported environments.

4. Suggestion and Explanation Engine

- Based on the identified errors or incomplete code, the system provides intelligent suggestions, such as code completions, bug fixes, and optimizations.
- Simultaneously, the NLP engine generates user-friendly explanations for each issue, with optional learning resources linked to the topic.

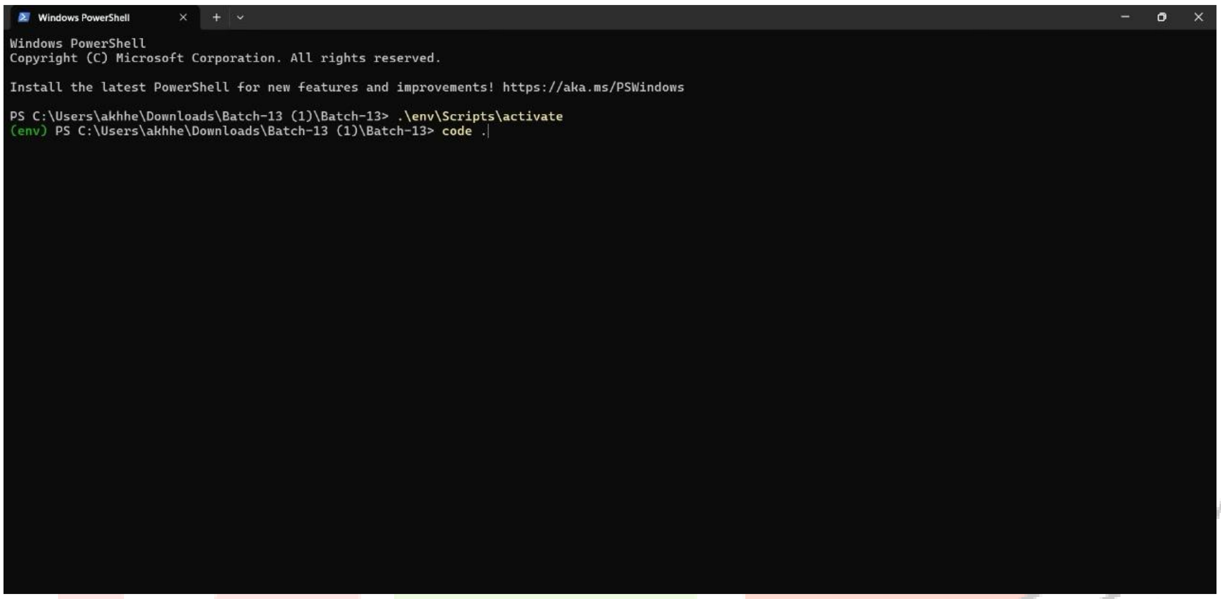
5. User Feedback and Adaptation

- The system tracks user responses to suggestions and explanations to improve future interactions.
- User profiles are updated to adapt the feedback level (basic to advanced) and recommend personalized content.

6. Integration with IDE

- The assistant is deployed as a plugin or extension that integrates with popular IDEs.
- It interacts with the IDE's API to access code context and deliver real-time feedback without interrupting the workflow.

RESULTS&ANALYSIS

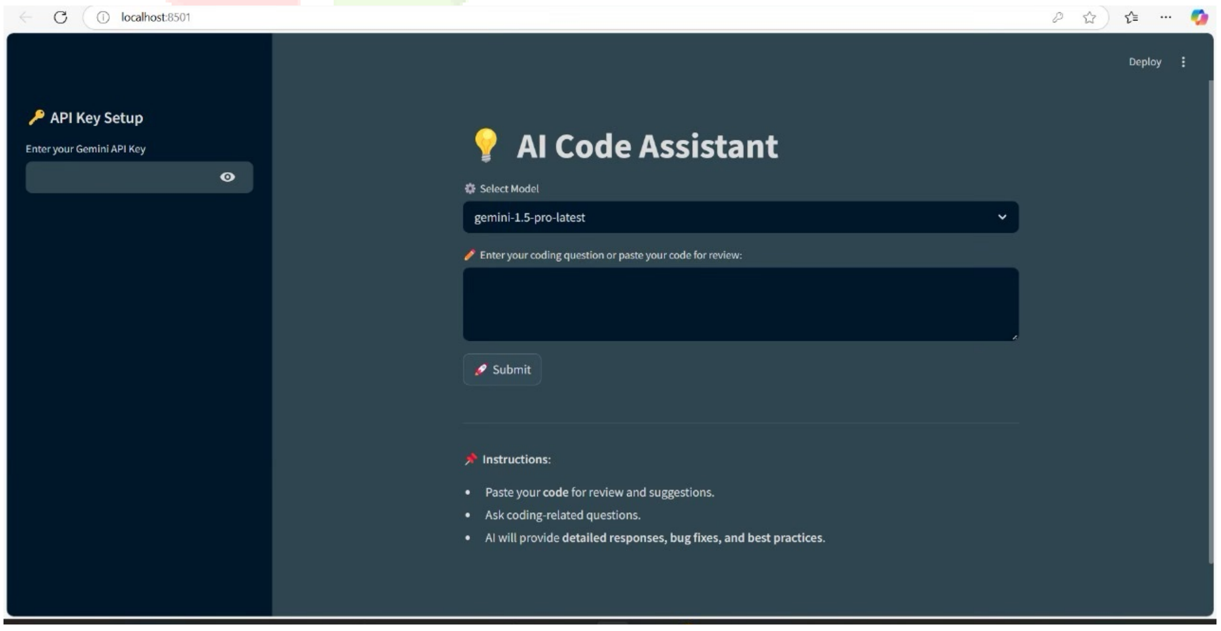


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\akhhe\Downloads\Batch-13 (1)\Batch-13> .\env\Scripts\activate
(env) PS C:\Users\akhhe\Downloads\Batch-13 (1)\Batch-13> code .
```

Figure1:command prompt



API Key Setup

Enter your Gemini API Key

AI Code Assistant

Select Model: gemini-1.5-pro-latest

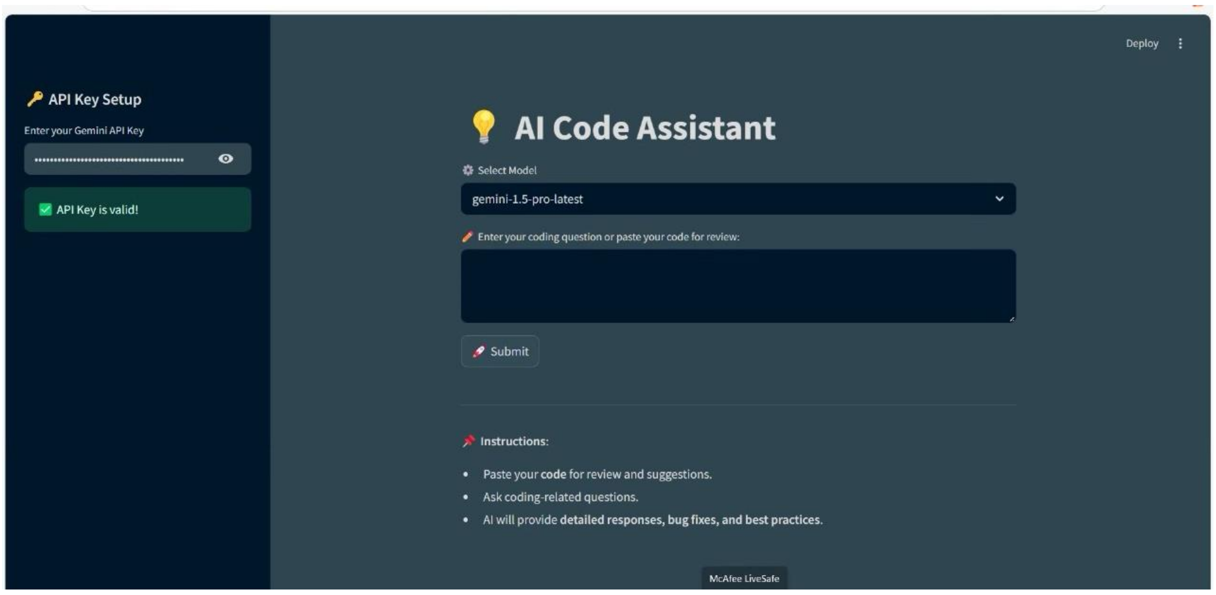
Enter your coding question or paste your code for review:

Submit

Instructions:

- Paste your **code** for review and suggestions.
- Ask coding-related questions.
- AI will provide **detailed responses, bug fixes, and best practices.**

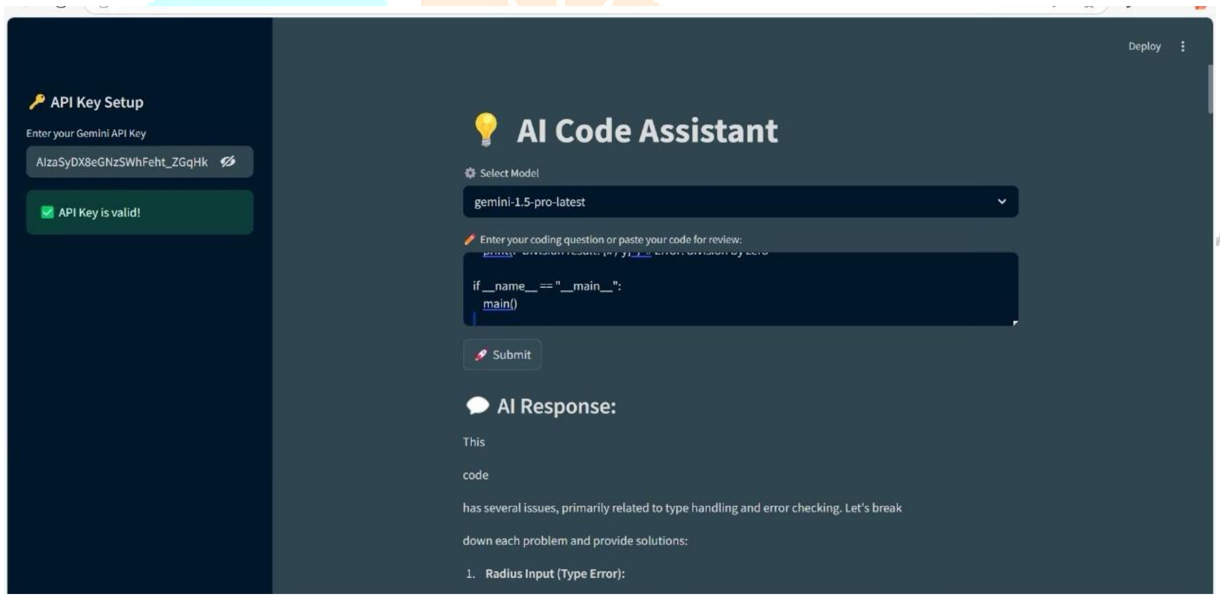
Figure 2:Home Page



Instructions:

- Paste your **code** for review and suggestions.
- Ask coding-related questions.
- AI will provide **detailed responses, bug fixes, and best practices.**

Figure3:Key Validation


Deploy
API Key Setup
Enter your Gemini API Key
AIzaSyDX8eGNzSWhFeht_ZGqHk
API Key is valid!
AI Code Assistant
Select Model
gemini-1.5-pro-latest
Enter your coding question or paste your code for review:
Submit

```
if __name__ == "__main__":
    main()
```

AI Response:

This

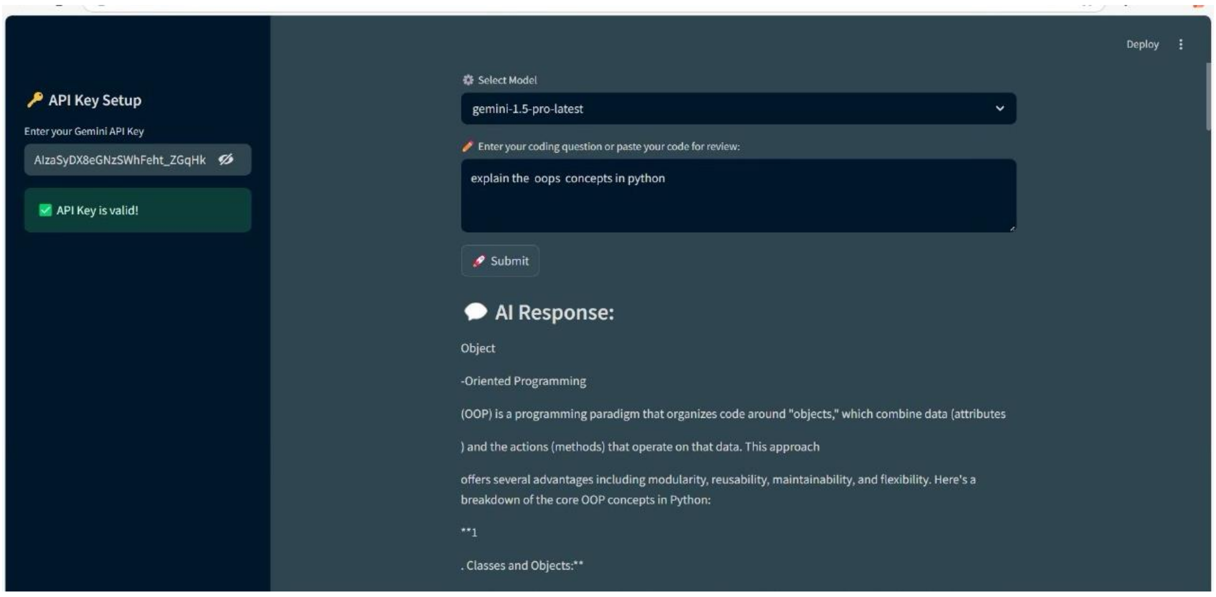
code

has several issues, primarily related to type handling and error checking. Let's break

down each problem and provide solutions:

1. **Radius Input (Type Error):**

Figure 4:code Debugging &Suggestions



Enter your Gemini API Key

AIzaSyDX8eGNzSWhFeht_ZGqHk

API Key is valid!

Select Model

gemini-1.5-pro-latest

Enter your coding question or paste your code for review:

explain the oops concepts in python

AI Response:

Object

-Oriented Programming

(OOP) is a programming paradigm that organizes code around "objects," which combine data (attributes

) and the actions (methods) that operate on that data. This approach

offers several advantages including modularity, reusability, maintainability, and flexibility. Here's a breakdown of the core OOP concepts in Python:

**1

. Classes and Objects:**

Figure 6:Learning Assistant

CONCLUSION:

This paper presents a novel AI-driven system designed to enhance the software development process by providing real-time code debugging, intelligent suggestions, and personalized learning assistance. By leveraging machine learning, natural language processing, and dynamic code analysis, the system not only improves developer productivity but also fosters an interactive and adaptive learning environment. The integration of these features into existing IDEs provides an intuitive, seamless experience that supports developers at all skill levels.

The proposed system addresses key challenges faced by developers, such as real-time error detection, context-aware suggestions, and learning in an uninterrupted development workflow. It bridges the gap between debugging tools and educational resources, making it an effective solution for both novice and experienced programmers. Additionally, through continuous user feedback and adaptive learning, the system evolves over time, offering increasingly personalized assistance.

Future work could explore expanding the system's language support, refining the accuracy of AI models, and incorporating additional features such as collaborative coding assistance. This intelligent assistant paves the way for a more efficient, accessible, and educational approach to coding, ultimately contributing to the improvement of software development practices.

REFRENCES

[1] Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M, February 13).The Impact of AI on Developer Productivity: Evidence from GitHubCopilot. arXiv.org. https://arxiv.org/abs/2302.06590.. (2023
[2] Sun, J., Liao, Q. V., Muller, M., Agarwal, M., Houde, S.,Talamadupula, K., & Weisz, J. D. February 10). InvestigatingExplainability of Generative AI for Code through Scenario-basedDesign. arXiv.org. https://arxiv.org/abs/2202.04903. (2022,
[3] Chen,M.,Tworek,J.,Jun,H.,Yuan,Q.,DeOliveiraPinto,H.P., Kaplan,J.,Edwards,H.,Burda,Y.,Joseph,N.,Brockman,G.,Ray,A.,Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P.,Chan,B.,Gray,s., Zaremba, W. Evaluating Large LanguageModelsTrainedonCode.arXiv.org.https://arxiv.org/abs/2107.03374,(2021, July 7).
[4] Kim,S.,Zhao,J.,Tian,Y.,&Chandra,S.CodePredictionbyFeedingTreestoTransformers.IEEE.https://doi.org/10.1109/icse43902.2021.00026,(2021).
[5] Lachaux,M., Roziere, B.,Chanussot,L., & Lample, G. UnsupervisedTranslation of Programming

Languages. arXiv.org.https://arxiv.org/abs/2006.03511, (2020, June 5).
[6] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L.,Qin, B., Liu, T., Jiang, D., & Zhou, M. CodeBERT: A Pre-TrainedModel for Programming and Natural Languages. arXiv.org.https://arxiv.org/abs/2002.08155,(2020, February 19).
[7] GitHub Copilot · Your AI pair programmer. (2024). GitHub.https://github.com/features/copilot/
[8] Duarte, F. J. F. Using AI as a development accelerator.https://recipp.ipp.pt/handle/10400.22/25979,(2024b, July 23).
[9] Semenkin, Anton, et al. "Full Line Code Completion: Bringing AI toDesktop."arXivpreprintarXiv:2405.08704 (2024).
[10] Nygård, Joonas.AI-assistedcodegenerationtools. MS thesis. J.Nygård, 2024.
[11] The Future of Software Engineering in an AI-Driven World. (n.d.).https://arxiv.org/html/2406.07737v1
[12] The Role of Generative AI in Software Development Productivity: APilot Case Study. (n.d.). https://arxiv.org/html/2406.00560v1
[13] Sergeyuk,Agnia,SergeyTitov,andMalihehIzadi."In-IDEHuman-AIExperience in the Era of Large Language Models; A LiteratureReview." Proceedings of the 1st ACM/IEEE Workshop on IntegratedDevelopment Environments. 2024.
[14] Nazari,AliReza,andBowNannichaThunell."UsageofGenerativeAIBased Plugin in Unit Testing: Evaluating the Trustworthiness ofGeneratedTestCasesbyCodiumate,anIDEPluginPoweredbyGPT-3.5 &4."(2024).
[15] Taylor, M., & Stevens, R. Leveraging AI to Improve DeveloperEfficiencyinIDEs.ComputationalSoftwareSystemsReview.(2022).