

**MANIPAL UNIVERSITY JAIPUR**

Directorate of Research

# Invention Disclosure Form – Confidential

Form-B

**Please Note: Kindly Fill this “form -B and Send the soft copy to manish.rawat@jaipur.manipal.edu**

|  |  |
| --- | --- |
| **DoR Reference Number:** |  |
| **Date of Submission:** |  |
| **MUJ Filing Categories-A, B, C** |  |
| **In Case Category-C selected,** Provide Filing Country Name |  |
| **Stage of Project: (Select one)**  Ideation  Ongoing work  Prototype  Working model  Results Validated | **Prototype** |
| **Submitted by Faculty: (Corresponding Inventor)**  Name, Designation  Department and School | **Mr. Vijay Prakash Sharma** |
| **Name of all Inventors**  Name, Designation  Department and School | **1.Aniruddhan Srinivasan**  **2.Manav Singh** |

A Patent Application must meet the following criteria in order to apply for patent:

**New**: The work must be new and have not been disclosed earlier, anywhere, in any form.

**Useful**: The work should generally be useful to company/industry/society and not merely for additional research/promotion.

**Unobvious**: Inventors should have identified unique aspects of the work in their research domain and should not be obvious to persons ordinarily skilled in the art.

1. **Proposed Title of the Invention**

AI-Assisted Code Review and Explanation Tool Using Pylint and LLM Integration - **BugBot**

1. **Proposed Abstract of the Invention**

An intelligent web-based platform that enables users to upload Python code files and obtain comprehensive diagnostics is introduced by this invention. The platform uses a big language model, such as CodeLlama, via the Ollama backend to deliver human-friendly explanations and fixes, and Pylint to identify syntax and style mistakes. The application shows contextual code samples to assist developers in improving code quality and includes a history system that lets users examine previous studies. With its automatic, perceptive response, this innovation expedites code review procedures, particularly in industrial and educational contexts.

1. **Key Words:**

AI, Code Review, LLM, Pylint, Python, Ollama, Bug Detection, Web Application

1. **Background of the Invention:**

Existing code analysis tools, such as Linters (like Pylint), identify problems but don't provide semantic justifications. Current systems frequently lack AI-powered suggestions and user-friendly interfaces. The majority also need technical setup or are desktop-based.

1. **What problems does the invention address and how your Invention is able to overcome the limitations/ problems of the existing technologies?**

This innovation combines static analysis with AI-driven explanations to overcome the drawbacks of conventional lanterns. Through straightforward web interaction, it helps people with little programming experience to comprehend and correct their mistakes. Additionally, traceability and learning over time are provided via the historical function.

1. **Detailed Explanation of the Invention along with working examples.**

Using a browser interface, users upload .py files to the Flask-based online application. Pylint analyzes the file and provides structured error/warning info. Some errors are sent to a local LLM (via Ollama) for real-time investigation, while others are matched with explanations. Code samples, best practices, error type, and fix technique are all explained. History is shown via a secondary channel and saved in a JSON log. Bootstrap is used to style the user interface

for responsiveness and accessibility.

1. **Please explain what is novel about the invention?**

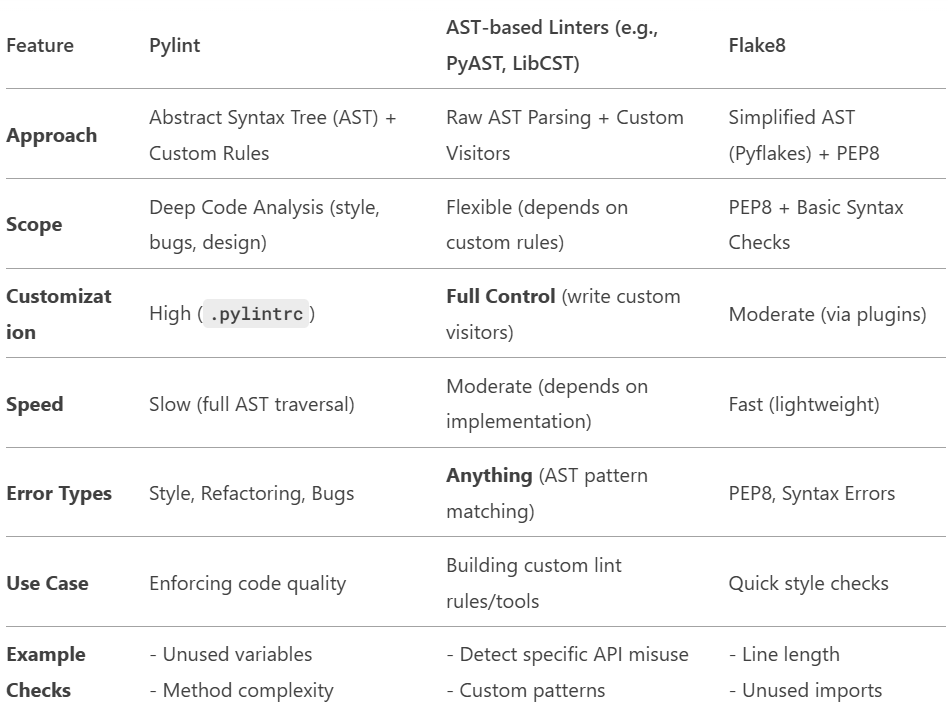
The novelty lies in merging static analysis (Pylint) with dynamic LLM-based explanations and an interactive browser-based UI. This approach transforms traditional linting into an educational and accessible experience, not obvious to standard development tools.

1. **What is the commercial viability of the innovation or is it capable of industrial production?**

The tool is applicable in educational institutes, developer onboarding, code quality audits, and debugging support platforms. It is lightweight, locally deployable, and does not require cloud connectivity, making it viable for commercial use.

1. **Kindly attach drawings, reports, papers, charts or other materials that may aid in your description.**

**Comparison of linting libraries we used to lint code:**



**Comparison of LLMs we used to achieve maximum speed and accuracy:**

A graph with a red line

AI-generated content may be incorrect.

**A view of what happens after a file is chosen for code review/analysis:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Overview of analysis history:**

A screenshot of a chat

AI-generated content may be incorrect.

1. **What are the aspects of your disclosure that you want to claim/monopolize?**

**Proposed Claims:**

**Claim 1: A code analysis platform that merges static analysis with LLM-based explanations**

This invention presents a hybrid code review system that effectively combines conventional static analysis tools (like Pylint) with large language models (LLMs) such as CodeLlama. The static analyzer identifies syntax errors, style issues, and code smells, while the LLM interprets the raw error messages and produces:

- Clear explanations of the errors for humans

- Step-by-step solutions to resolve the issues

- Best practices to prevent similar errors in the future

This two-tiered architecture improves the system's diagnostic capabilities, allowing both beginner and experienced programmers to learn from and correct errors more efficiently.

**Claim 2: A web interface for file uploads, AI-driven diagnosis, and history tracking**

The system features a user-friendly, web-based front end developed with modern web technologies (Flask, HTML/CSS, and Bootstrap). Users can easily upload .py files using a drag-and-drop feature or a form-based interface. After uploading:

- The backend analyzes the file with Pylint

- The detected issues are sent to the AI engine (LLM) for contextual explanations

- Results are shown in real-time, with an option to view highlighted code snippets

Moreover, each analysis session is recorded and stored in a persistent history file (history.json). Users can access the "View History" page to review all past file analyses, complete with timestamped records, error summaries, and AI-generated insights, promoting a continuous learning process and audit trail.

**Claim 3: Integration of user-friendly code context and AI-generated best practices for enhancement**

In addition to standard linters that provide line numbers and brief warnings, this invention offers:

- Code context extraction: Automatically retrieves and displays 2–3 lines before and after the problematic line, providing a clear view of the issue.

- LLM-generated recommendations: Rather than merely flagging problems, the AI provides:

- Clear explanations of why the code is incorrect

- Suggestions for how to rewrite or refactor the code segment

- General programming principles or Python best practices related to the issue

This integrated feedback system not only addresses errors but also educates users, effectively turning the tool into a smart teaching assistant.

1. **Have you conducted novelty/inventiveness search for your invention? If yes, what are the databases /references used by you? What Keywords did you use and What are the search results?**

**US10635409B2 – System and Method for Enhancing Software Code Quality Using AI Techniques**Summary: This patent details a system that gathers training data from source control systems and integrated development environments (IDEs) to train an artificial neural network. The resulting model offers suggestions to enhance the quality of new or altered software code by detecting violations of coding standards and recommending fixes.  
Key Features:

* Extraction of training data from SCM systems and IDEs
* Training of a machine learning model
* Recommendation module for improving code quality
* Remediation module for addressing coding standard violations  
  Link: US10635409B2

**US10877869B2 – Method and System for Developing a Code Review Tool**

Summary: This patent describes a code review tool that evaluates software code against established standards and requirements. It features modules for standards and requirements, an analysis processor, and a user interface to present the results of the code review.  
Key Features:

* Module for standards and requirements
* Analysis processor for evaluating code
* User interface for displaying code review results  
  Link: US10877869B2

**US20190228319A1 – Data-Driven Automatic Code Review**

Summary: This application presents a system that employs deep learning models, including neural networks, to conduct automatic code reviews. The system evaluates code changes and offers feedback based on patterns learned from past code reviews.  
Key Features:

* Code assessment based on deep learning
* Use of historical code review data
* Automatic generation of feedback for code modifications  
  Link: US20190228319A1

**What makes our application different?**

- **Integration of Static Analysis and LLM:** Your system uniquely combines traditional static code analysis (utilizing tools like Pylint) with the capabilities of large language models (LLM) to deliver thorough explanations for coding problems.

- **Web-Based Interface:** In contrast to the mentioned patents, your tool features an intuitive web interface that enables users to upload Python files, view analysis results, and access past data.

- **Educational Emphasis:** By providing detailed explanations and best practices, the tool aims to educate users, making it particularly useful for learners and developers who want to understand and fix coding issues.

- **Offline Capability:** Your system is designed to function without requiring cloud connectivity, making it suitable for use in areas with limited internet access.

1. **Do you feel that a person of “average” skill (not-extraordinary skill) in your area of technology would have arrived at your invention with existing knowledge in public domain? If no, what could be the reasons for the same?**

**No,** an average-skilled individual in the field would not have easily developed this invention using only publicly available knowledge.

**Reasons:**

**1. Complexity of Integration:** The invention skillfully merges several areas—static code analysis, AI/LLM processing, natural language generation, and web development—requiring a level of interdisciplinary expertise that is not typically found in an average developer or student.

**2. Non-obvious Application:** Although Pylint and LLMs are available separately, it is not straightforward to integrate them into a cohesive educational tool that clarifies errors, offers best practices, and presents results through an engaging web interface.

**3. Focus on Educational Utility:** The emphasis on transforming a code linter into a learning platform with detailed error breakdowns and AI-driven explanations reflects a creative approach to user-centered design, which is uncommon in standard linting or debugging tools.

**4. Local LLM Integration**: Unlike most tools that depend on cloud-based APIs like OpenAI, this invention operates locally with Ollama and CodeLlama, which minimizes latency, enhances privacy, and improves accessibility—features that are not typically found in existing solutions.

1. **Kindly provide broad workable ranges for all the parameters involved in your invention.**

The invention primarily emphasizes software functionality, but several important operational parameters shape its deployment environment, usability, and system constraints. These parameters are crucial for the efficient, scalable, and user-friendly operation of the code quality enhancement system:

**Supported File Type:** The system is specifically designed to analyse Python source code files with the .py extension. This focused approach allows for enhanced optimization and better management of Python-specific syntax, idioms, and coding standards. While support for other programming languages may be explored in future updates, it is not currently included.

**File Size for Analysis:** The analysis engine is optimized for Python files that range from 1 KB to about 500 KB, which typically equates to around 1000 lines of code. This size range ensures that both small scripts and moderately large modules can be processed effectively without straining the system or surpassing LLM processing limits. Files larger than this may need to be divided or processed in batches for thorough review.

**Number of Errors Processed per File:** The system can identify and manage a wide range of code issues per file, from none (for clean code) to over 100 distinct errors or warnings. These issues may include breaches of coding conventions, potential bugs, inefficiencies, and anti-patterns. The system is designed to intelligently scale its analysis based on the density and severity of the identified issues.

**LLM Response Timeout:** Each explanation or recommendation produced by the integrated large language model (LLM) is subject to a configurable timeout, typically set between 5 to 60 seconds. This range strikes a balance between providing detailed, context-aware responses and maintaining responsiveness for user experience. For more complex code or bulk reviews, longer timeouts may be beneficial, while quicker checks can be done within shorter limits.

**User Interaction Modes:** The system accommodates various modes of user interaction to meet different deployment requirements:

* Single-user local deployment: This mode is perfect for individual developers or offline use, allowing users to analyse code on their local machines without needing a network connection.
* Multi-user web-based deployment: This mode facilitates centralized, session-managed access for multiple users through a web interface, making it suitable for teams or organizations that require collaborative code review capabilities and shared review histories.

**History Retention:** The system provides flexible history retention options. Depending on the available storage and configuration, it can retain anywhere from no session memory to an unlimited number of past code review sessions. This feature allows users to revisit previous analyses, monitor improvement trends, and maintain continuity in long-term projects.

1. **References**:

**1.1 "Tufano et al.'s "Learning Bug-Fixing Patches in the Wild using Neural Machine Translation"**

**Focus:** Learns bug-fix patterns from GitHub Java commits using Neural Machine Translation (NMT).

**Method:** Generates patches by extracting method-level bug-fix pairs (BFPs), abstracting code tokens, and training an encoder-decoder model.

**Key Findings:** 82–99% of generated patches are syntactically valid; 9–50% accuracy in predicting developer-like modifications is achieved.

**1.2(Sejfia et al., ESEC/FSE '21) "Identifying Casualty Changes in Software Patches"**

**Focus:** Introducing casualty changes, which are unnecessary patch adjustments that result from other changes rather than changing the program's logic.

**Method:** Introduces CasCADE, an automated detector that uses AST and static analysis, along with a taxonomy (API-based, variable-based, and refactoring-based).

**Key Findings:** Casualty modifications are present in 21% of security patches. Noise reduction increases the accuracy of patch-based tools (like Assoc Checker) by 18%.

1. **List Names and details of All Inventors (Full Names, Nationality and Addresses)**

**Inventor 1:**

Name: Aniruddhan Srinivasan

Designation: Student

Department: Information Technology

Contact Details: 8939582446, aniruddhsrinivasan@gmail.com

Nationality: Indian

**Inventor 2:**

Name: Manav Singh

Designation: Student

Department: Information Technology

Contact Details: 6207492015, manavs983@gmail.com

Nationality: Indian

1. **Applicant Details**
2. **Aniruddhan Srinivasan – 229302128, B.Tech Information Technology**

**Address: 20/21 Luz Avenue 3rd street, Mylapore, Chennai-600004**

1. **Manav Singh – 229302140, B.tech Information Technology**

**Address: Singh More, Vikas Nagar, Ranchi-834003**

***Manipal University Jaipur will be the Applicant for all IPR filed by MUJ Faculty and Staff***

1. **Any additional notes or remarks.**

This invention possesses considerable educational implications, particularly for higher education institutions, coding bootcamps, and individuals engaged in self-directed learning. By providing real-time, AI-generated explanations for coding errors, it transforms the typically challenging debugging process into an interactive educational experience.

The tool is engineered to function entirely offline, requiring no internet connectivity once the necessary dependencies are installed. This feature renders it particularly suitable for secure, low-bandwidth, or restricted environments, such as examination laboratories, corporate firewalls, or rural educational settings.

Prospective developments may encompass support for multiple programming languages, integration with version control systems, and the creation of role-specific dashboards tailored for educators, learners, and developers.

The invention exhibits novelty not only through its technical integration but also in its redefinition of the role of code linting—transitioning from a focus on error detection to an emphasis on error education.