AI-Assisted Python Code Quality Analyzer

Intelligent Code Review Powered by AI and Static Analysis

Automatically detect code problems, get clear explanations, and fix issues safely. Integrates seamlessly into your development workflow with CLI, Git hooks, and CI/CD support.

Presented By:

Name : Sahasra Doddi

The Code Quality Conundrum

Good code quality is important, but manual reviews are slow and inconsistent. Feedback gets scattered across different tools and messages, leading to varied application of standards. Important issues are often missed, while older code checking tools highlight problems without clear explanations or actionable fix suggestions.

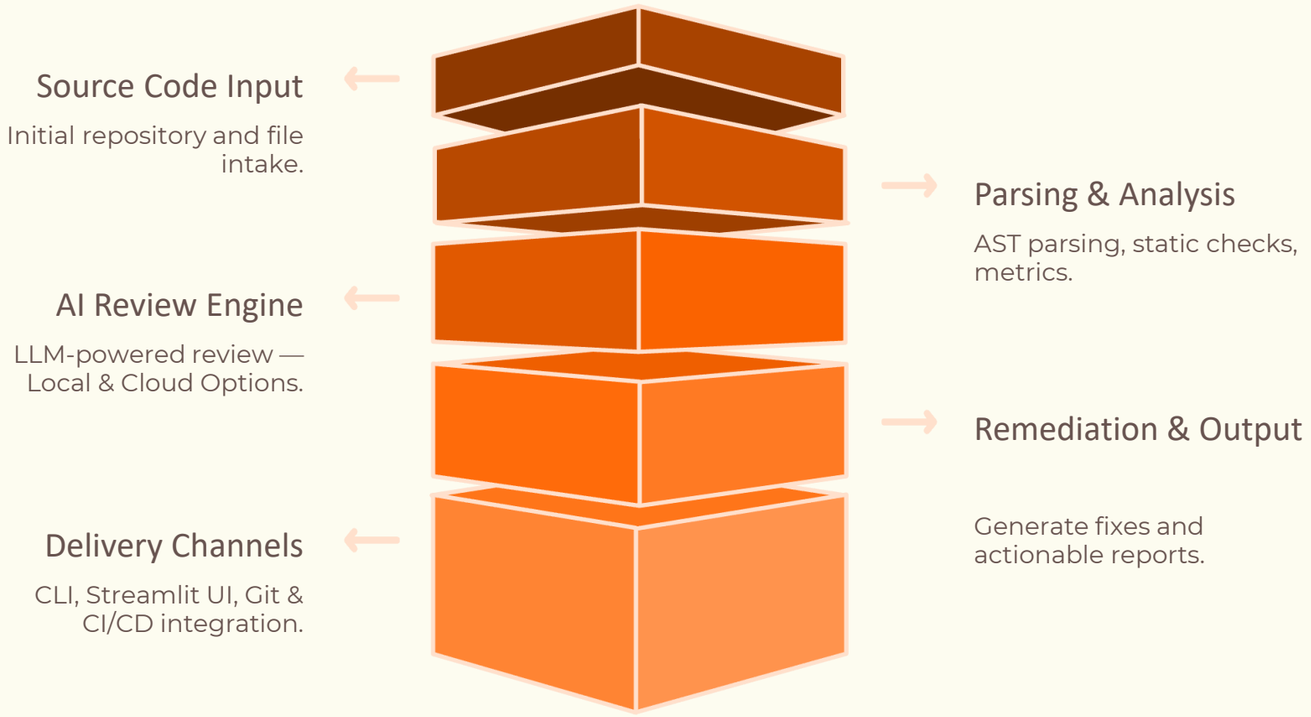
We need a smarter approach: a single tool that automatically finds code problems, explains them clearly, and suggests exact fixes. This helps developers work faster, ensures consistent quality, and prevents costly problems before they escalate. It also improves team understanding and overall development efficiency.

Objectives

- Automate Python code analysis for quality issues.
- Leverage AI and static analysis for smart explanations. explanations.
- Deliver clear quality metrics and reports.
- Enable safe, automated code fixes (AutoFix).
- Integrate seamlessly into developer workflows.
- Support flexible configuration and extensibility.

System Architecture

Our system operates through four distinct layers that work together to provide end-to-end code quality analysis. First, the **Input Layer** takes code files directly from your project. This then moves to the **Analysis Layer**, where static analysis and Abstract Syntax Tree (AST) parsing are performed to identify potential issues. Next, the **AI Layer** leverages advanced AI engines to provide clear explanations and intelligent suggestions for found issues. Finally, the **Output Layer** generates comprehensive reports, key metrics, and offers automated fixes, integrating seamlessly back into your workflow.



Data Flow

Code Input & Configuration

Python files are scanned and project configurations are loaded.

Static Analysis

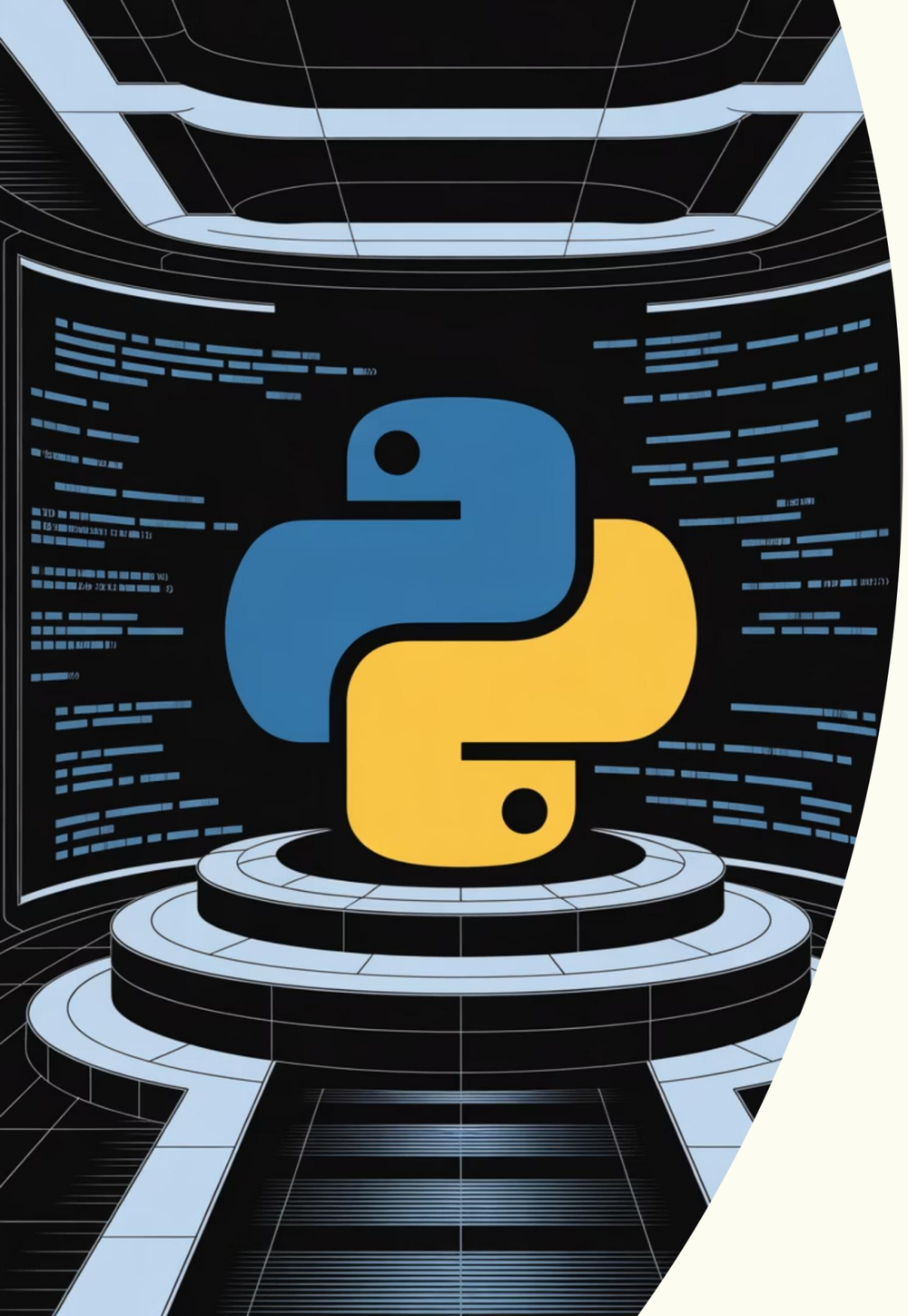
Static analysis parses code and detects issues.

AI Review & Suggestions

An AI engine explains problems and suggests fixes.

Output, Reports & Fixes

Reports and automated fixes are generated.



1.Core Analysis Layer

The **Core Analysis Layer**, powered by our **CodeQualityAnalyzer**, is where your code undergoes a thorough examination. It uses **Abstract Syntax Tree (AST) parsing**, which means it breaks down your code into its fundamental structure, understanding its logic and relationships. This allows it to precisely detect a wide range of **code smells**, **style issues**, and **maintainability problems**. For instance, it identifies common issues like **long methods**, **god classes**, **deep nesting**, **unused imports**, **missing type hints**, and **missing docstrings**. Beyond identification, it also measures critical quality metrics such as **code complexity**, **lines of code (LOC)**, and a comprehensive **maintainability index**, providing a clear picture of your codebase's health.

Smell Detection

Long methods, god classes, deep nesting, unused imports, missing type hints & docstrings

Quality Metrics

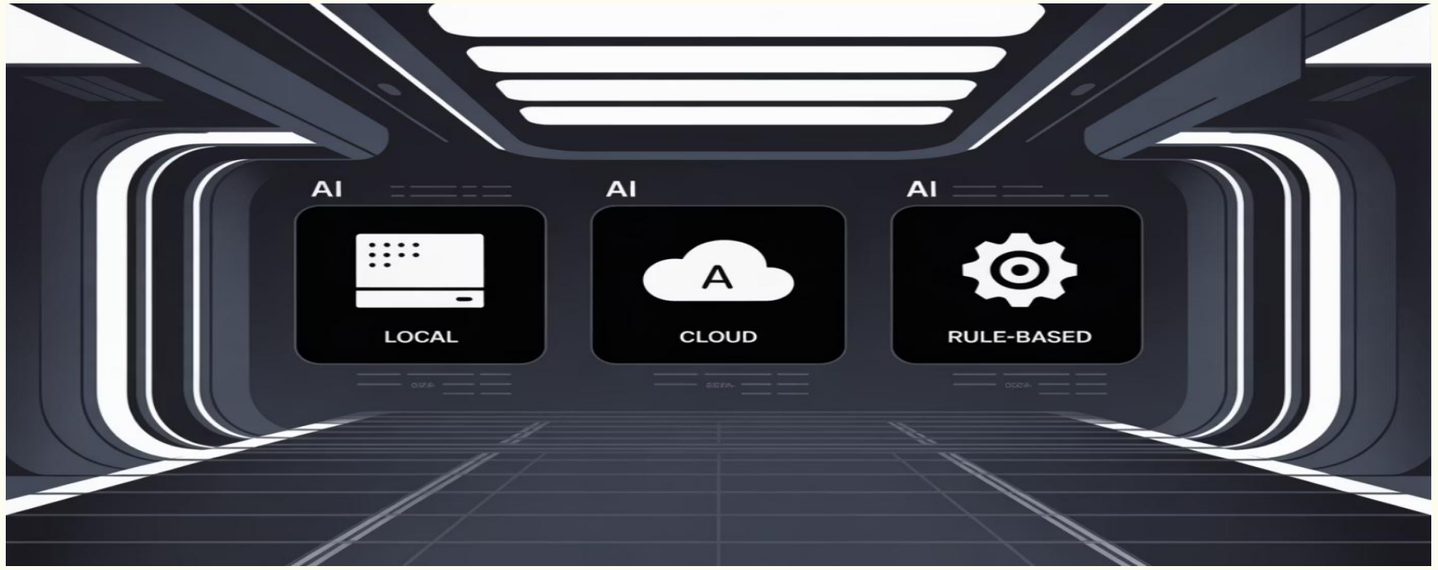
LOC, Maintainability Index, per-file and project-level quality scores

Report Output

Structured JSON and CSV exports for downstream consumption

2. AI Review Engine

Our system intelligently combines three distinct AI review engines for comprehensive and reliable code analysis. They act as a sophisticated fallback, providing detailed explanations, severity ratings, and improvement suggestions.



Ollama (Local)

- Phi-3 Mini locally
- Private, no internet
- Fast analysis

OpenRouter (Cloud)

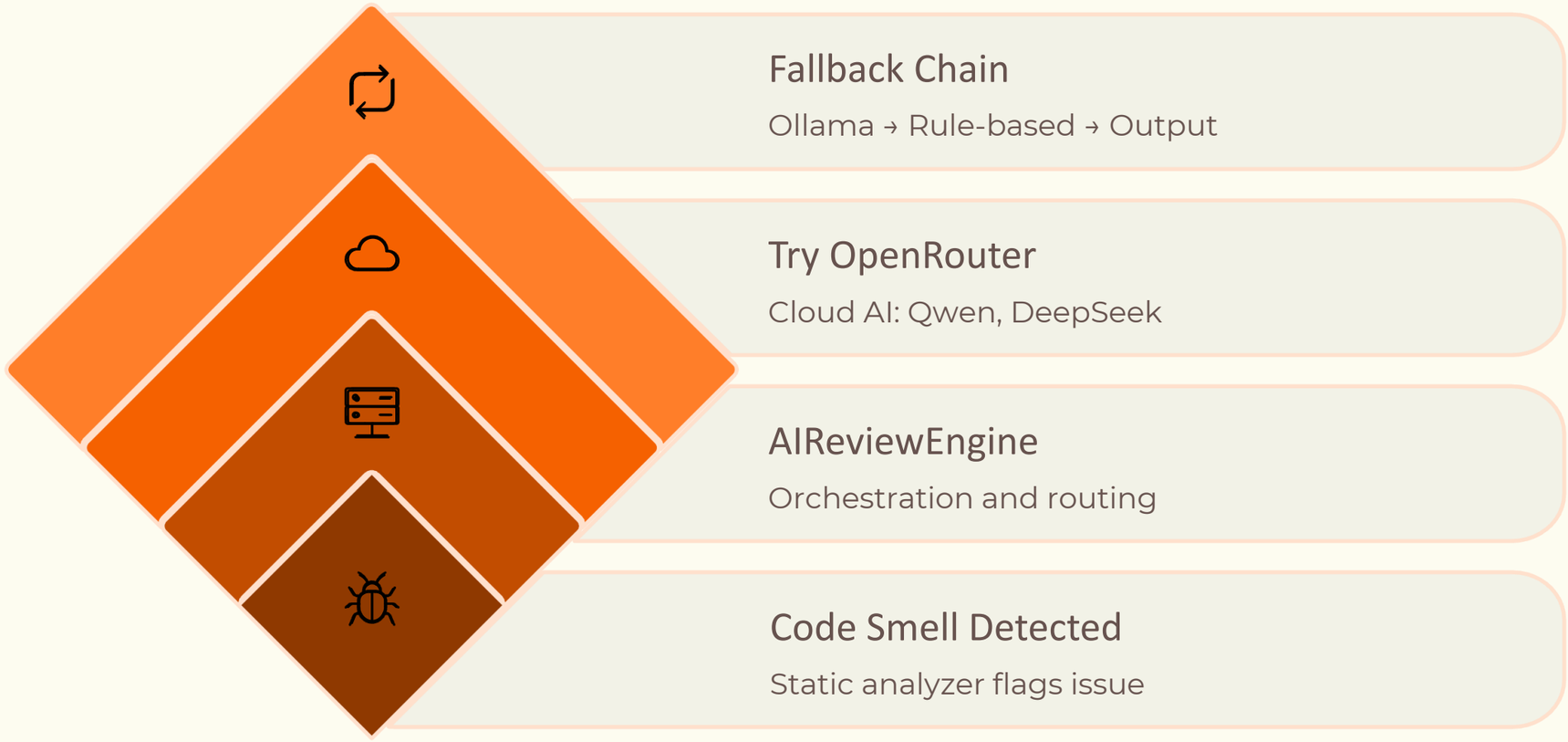
- Qwen, DeepSeek, Gemma etc
- Powerful analysis
- Internet required

Rule-Based (Fallback)

- Predefined templates
- Always works
- No AI needed

AI Review Engine - Fallback Chain

Our AI Review Engine is designed with a robust fallback system to ensure continuous and reliable code quality analysis. When a code smell is detected by the static analyzer, the system intelligently attempts to process it through a hierarchy of AI engines, starting with powerful cloud models, then local AI, and finally a dependable rule-based engine, guaranteeing a comprehensive review comment every time.



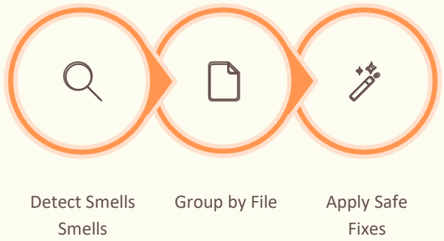
Automation & Remediation

AutoFixEngine

The AutoFixEngine automatically fixes simple, safe code issues while preserving behavior. It groups problems by file and applies line-level corrections.

Key Features:

- **Preserves code behavior** - Only fixes simple, well-defined issues
- **Transparent tracking** - Records original code, fixed code, and reason
- **Developer control** - All fixes presented for review before applying
- **Safe by design** - Complex issues always require human judgment



Supported Fixes:

- Remove unused imports and variables
- Add missing docstrings
- Fix naming convention violations
- Correct style inconsistencies
- Add obvious type hints

Each `AutoFix` record includes: file path, line number, original/corrected code, status, and explanation.

3. Validation & Metrics Layer

Code Smell Detection

Uses Python's AST module to parse source files and identify issues like long methods, god classes, deep nesting, unused imports, missing type hints/docstrings

File-level Metrics

Computes Lines of Code (LOC), Maintainability Index (MI), and assigns quality scores based on smell count and severity weights

Project-level Aggregation

Recursively analyzes all Python files, aggregates metrics, calculates docstring coverage percentage, and generates comprehensive project statistics

Structured Reporting

Generates project_quality_report.json, detailed_smells.csv, and HTML dashboards for easy consumption

Validation Logic

Severity classification, quality scoring (0-10 scale), project quality gates (fails if avg_quality_score < 6.0)

Key Output

Actionable, quantifiable quality signals that drive gates, dashboards, and decision-making

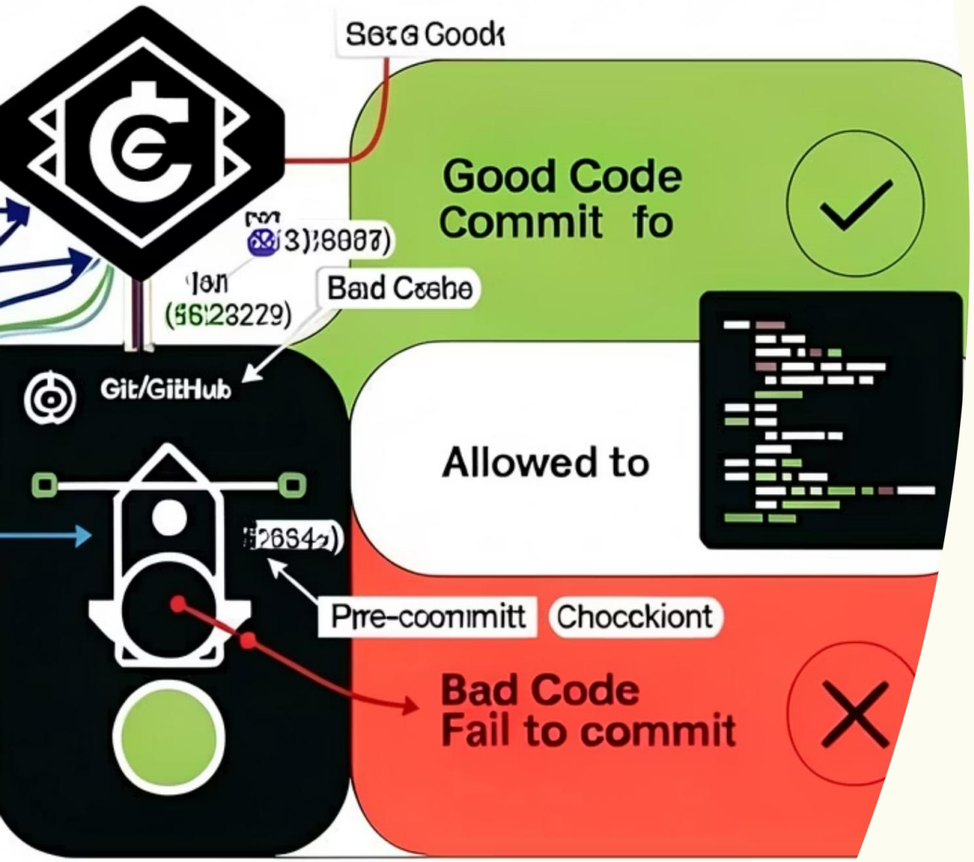
4. Command-Line & CONFIGURATION

User-friendly, Click-based CLI:

- **scan**: Initiates a comprehensive check of your codebase for potential issues and coding standard violations. It's useful for quick checks and integrating into CI/CD pipelines.
- **review**: Leverages our advanced AI engine to provide in-depth explanations of detected problems and offers intelligent suggestions for improvement. This helps developers understand the root cause and accelerate the learning curve.
- **autofix**: Automatically applies recommended fixes directly to your codebase, resolving common issues without manual intervention. This reduces time spent on routine code corrections and ensures code quality.
- **report**: Generates comprehensive quality reports detailing identified issues, their severities, and actions taken. These reports are invaluable for tracking code quality metrics over time.
- **diff**: Provides a clear visualization of proposed changes before they are applied, allowing you to review the impact of automated fixes. This ensures transparency and control over codebase modifications.

Project Configuration

- **pyproject.toml:** Defines global settings like quality thresholds and preferred AI models.
  - Specifies project metadata and dependencies for consistent builds.
  - Manages build system configuration and tool-specific settings.
- **settings.json:** Customizes code smell types and automatic fix rules.
  - Enables or disables specific code checks to tailor analysis.
  - Configures severity levels for different issues based on project needs.
  - Sets up ignore patterns for files or directories to exclude from analysis.
- **.env:** Securely manages API keys and sensitive credentials.
  - Facilitates environment-specific configurations without hardcoding.
  - Keeps sensitive information out of version control for enhanced security.



5. VCS & CI Integration

Git Pre-commit Integration

- Integrate quality checks directly into your local development workflow using Git pre-commit hooks.
- Prevent low-quality code commits at the source.
- Use `ai-gate-single` for focused feedback on individual files.
- Use `ai-gate-batch` for comprehensive analysis of all staged changes.
- Block commits that don't meet predefined quality thresholds.

GitHub Actions CI Workflow

Ensure consistent code quality across your codebase.

- Act as a final quality gate for all pull requests.
- The `code-quality.yml` workflow includes scanning, optional auto-fixes, and critical quality gates.
- Automatically add review comments to pull requests.
- Block merges if the average quality score falls below a predefined threshold.

This layered defense system, combining Git pre-commit hooks and GitHub Actions CI, provides robust code quality assurance by ensuring only high-quality, compliant code merges into your main branches.

6. REVIEW WEB UI

Ship Cleaner, Safer Python — Faster

Streamlit dashboard that scans Python code, spots quality and security issues, delivers AI-written feedback, and applies safe auto-fixes in one workflow.

Scan Python files or entire projects to find bugs, security exposures, lint errors, and performance hotspots. Get clear, prioritized AI feedback and preview safe, reversible auto-fixes in a sandboxed workflow.

Built for engineers and dev teams who want faster reviews, fewer regressions, and less manual patching—centralized scanning, explainable AI guidance, and one-click fixes that speed merges while keeping developers in control.

How It Works

A clear, 5-step workflow that shows how the Streamlit code reviewer scans a project, analyzes modules, runs AI reviews, and unlocks actionable tabs for developers.

1) App Starts

The Streamlit app boots: UI loads, extensions initialize, and the review engine becomes ready to accept input.

2) Module Loading

Project modules are discovered. Shows what's cached (reused) vs optional modules to load. Fast incremental analysis when cache available.

3) Sidebar Input

User uploads a file or project, toggles analysis engines (linters, security, perf, AI), and sets thresholds for warnings and autofix behavior.

4) Run Analysis

The app creates a temporary sandbox directory, runs static analysis and tests, and computes metrics (issues, severity, coverage, hotspots).

5) Tabs Unlock

Results populate five interactive tabs: Overview, Smells, AI Review, Auto-fix, and Reports — each provides insights, explanations, and actions.

This sequential flow keeps scans fast, results explainable, and fixes reversible — so teams move faster with confidence.

Key Features

The dashboard provides a comprehensive code quality analysis workflow with five integrated tabs, each designed to help developers identify, understand, and fix code issues efficiently.

Module-aware Loader

Safely imports `CodeQualityAnalyzer`, `AutoFixEngine`, and multiple review engines while showing environment status and dependency health.

Overview Tab

Displays project-level metrics including average quality score, total smells, severity distribution, and per-file analysis with quality gates.

Smells Tab

Lists all detected `CodeSmell` instances with filters for severity and type to drill into specific issues.

AI Review Tab

Runs AI-generated reviews using Rule-based, OpenRouter, or Ollama engines and provides clear explanations and actionable suggestions.

Auto-fix & Reports Tabs

Applies safe and AI-powered fixes through the AutoFixEngine and exports metrics and findings as JSON and CSV for comprehensive reporting.

Technology Stack

Python 3 & Core Modules

Our system is built on **Python 3** for scripting and data manipulation. The built-in `ast` module is crucial for parsing Python code into an Abstract Syntax Tree, enabling detailed static analysis. We use `dataclasses` for efficient data structures and `pathlib` for object-oriented file system paths. Standard modules like `json` and `csv` handle report generation. `subprocess` manages external program execution, including interactions with various AI models.

LLM Integration

Our system integrates with both local and cloud-based Large Language Models (LLMs). We use **OpenRouter**, accessed via `requests`, to tap into various powerful cloud AI models. For local execution and privacy-sensitive operations, **Ollama** is used through `subprocess` calls. A robust custom rule-based fallback mechanism ensures consistent performance even when LLM services are unavailable or ambiguous.

Analysis Tools & CLI

For a seamless command-line experience, we integrate `click` for powerful CLI applications. We utilize static analysis tools like **Pylint** and **Flake8** to identify errors and style violations. **Radon** is employed to calculate code complexity metrics, providing insights into maintainability. Configuration is managed efficiently with `python-dotenv` for environment variables and `tomli/tomllib` for structured configuration files.

Planned: Streamlit UI

We're developing an optional web user interface using **Streamlit**. This interactive dashboard will provide a visual representation of detected issues, allowing users to easily navigate code quality reports. It will feature side-by-side diffs for clearer comparison and resolution of identified problems. This UI aims to offer a more intuitive way to interact with our analysis engine, with dependencies listed in `requirements.txt`.

Summary & Roadmap

✅ Currently Implemented

Our tool is ready to use with robust features including comprehensive Python code analysis and issue detection via Abstract Syntax Tree (AST), multiple AI review engines leveraging both local and cloud-based Large Language Models, and an AutoFix capability for simple, common issues. Users can interact with the tool through a powerful CLI interface, integrate it seamlessly into their development workflow using Git pre-commit hooks, and ensure continuous quality with CI integration. Detailed quality metrics and reports provide actionable insights into codebase health.

🔄 Planned Features

Looking ahead, we are actively developing a Streamlit-based web dashboard to provide a more intuitive visual interface for reviewing issues, comparing code changes with side-by-side diffs, and managing the accept/reject workflow. Future plans include expanding support to additional programming languages like Java, Go, and TypeScript, introducing advanced filtering and search capabilities for reports, and enabling team collaboration features. We also aim to offer enterprise integrations for larger organizations and streamline distribution as a standard Pip package for easy installation.

🎯 Our Goal

Enforce code quality automatically at every stage of the development lifecycle, providing developers and teams with the tools needed to maintain high standards efficiently.

Our core system currently offers immediate value with its robust Python analysis, AI-powered reviews, and automated fixes, integrated directly into developer workflows. Coming soon, a new web dashboard will enhance visual interaction and management of code quality. The tool is designed to grow, extending its language support and enterprise features to become a comprehensive solution for maintaining high code standards across diverse development environments.

Comparison: Manual vs AI-Assisted Review

Speed

Manual reviews take days. AI analysis gives feedback in seconds.

Consistency

Different reviewers judge code differently. AI applies the same rules every time.

Accuracy

Humans miss problems. AI catches security issues and bugs before they go live.

Scalability

Manual reviews slow down as code grows. AI can handle any size project.

Future Enhancements

Support More Languages

Add support for Java, Go, TypeScript, and other popular programming languages.

Better Dashboard

Add team collaboration features, filters, alerts, and approval workflows to the dashboard.

Easy Installation

Make it easy to install and use with different tools. Add reports in PDF and Excel formats.

Conclusion

This AI Code Quality Analyzer changes how teams review code. Instead of slow manual reviews, it automatically checks code in seconds. It finds problems, explains them clearly, and suggests fixes. The tool works with your existing development tools and can grow with your team. It helps developers work faster while keeping code quality high and secure.

Thank You

AI-Assisted Python Code Quality Analyzer — Reliable, Secure, Future-Ready.