AI Code Judge

Overview

AI Code Judge is a sophisticated, AI-powered code analysis tool developed with Streamlit and LangChain. It leverages advanced large language models (LLMs) to deliver in-depth, automated code reviews for developers, educators, and code quality enthusiasts.

The tool analyzes code snippets across multiple programming languages, providing actionable insights on syntax, logic, best practices, security, performance, and maintainability.

What It Does

AI Code Judge acts as an intelligent code reviewer that goes beyond basic linting. It uses Groq's **Llama-3.1-8b-instant** model to perform static analysis, calculate metrics, detect potential issues, and suggest refactoring. Users can upload code files, paste snippets, or analyze entire GitHub repositories, making it useful for:

- Individual developers
- Code review workflows
- Educational purposes
- Team collaboration

How It Helps

- **Developers**: Receive instant feedback on code quality, identify bugs early, and learn best practices.
- **Educators**: Teach programming concepts with real-world analysis examples.
- **Teams**: Integrate into workflows to ensure consistency and catch issues before deployment.
- **Learners**: Understand structure, complexity, and areas of improvement through detailed reports.

The tool also promotes better coding habits with visual dashboards that quantify code quality.

Features

Core Analysis Capabilities

- Multi-language code analysis powered by LLMs
- Detection of syntax errors, logical issues, and best practice violations
- Identification of security vulnerabilities and performance bottlenecks

- Refactoring suggestions with clear recommendations
- Code smell detection (e.g., long methods, duplicate code)

Static Code Metrics Dashboard

- Cyclomatic Complexity
- Maintainability Index (0–100 scale)
- Lines of Code (LOC)
- Readability (Flesch-Kincaid grade level)
- Halstead Metrics (vocabulary, volume, difficulty, effort)
- Nesting depth, function count, variable count
- Duplication percentage
- Comment density and average function length

Metrics are displayed with **color-coded status indicators** for clarity (green = good, red = concerning).

Advanced Features

- Interactive AI-powered chat about code
- Persistent analysis history for tracking progress
- Side-by-side code comparison with diffs and metric differences
- Multi-file project-level analysis
- GitHub repository analysis by URL
- Export results to **PDF** or **JSON**
- Adjustable AI creativity/consistency via temperature settings

---

User Interface

- Professional, responsive Streamlit design
- Collapsible tabs for organized results
- Visualizations (bar charts, comparisons, dashboards)
- Wide layout optimized for desktop use

Architecture

Components

- **Frontend (app.py)**: Streamlit interface for user interaction
- **AI Engine (main.py)**: LangChain + Groq-based analysis
- **Utilities (utils.py)**: Metric calculations and code processing

- **Chat Module (chat.py)**: AI conversational analysis
- **Export Module (analysis_export.py)**: PDF and JSON reporting
- **Comparison Module (code_comparison.py)**: Side-by-side diff and metrics

Data Flow

1. User submits code
2. Static metrics are calculated
3. Code is sent to the AI model with structured prompts
4. Results are parsed, formatted, and displayed
5. Analysis history is stored in session state

Technology Stack

- **Frontend**: Streamlit
- **AI/ML**: LangChain + Groq API
- **Code Processing**: Black formatter (Python), regex-based checks
- **Visualization**: Streamlit charts and custom styling
- **Export**: ReportLab (PDF), JSON output

How It Works

Analysis Process

1. Input code via file upload or text input
2. Automatic language detection
3. Static analysis for metrics
4. AI review with structured prompts
5. Results organized into categories with visual indicators
6. Dashboards display insights

Prompt Engineering

Prompts instruct the AI to:

- Detect language and syntax errors
- Identify bugs and vulnerabilities
- Suggest best practices and refactoring
- Estimate complexity and test coverage
- Provide confidence scores

Metric Calculation

- **Cyclomatic Complexity**: Control flow paths
- **Halstead Metrics**: Operators and operands
- **Readability**: Flesch-Kincaid grade level
- **Code Smells**: Anti-pattern detection

Language Support

Supported Extensions

- Python (.py)
- JavaScript (.js), TypeScript (.ts)
- Java (.java), Kotlin (.kt)
- C (.c), C++ (.cpp)
- Rust (.rs), Go (.go)
- PHP (.php), Ruby (.rb)
- Swift (.swift)
- HTML, CSS, JSON, XML
- Plain text (.txt) for other languages

Limitations

- Formatting limited to Python (via Black)
- Metrics are heuristic and optimized for imperative languages
- No runtime execution; static and AI-based only

Installation

1. Clone the repository:
   ```
   git clone https://github.com/yourusername/code-judge.git
   cd code-judge
   ```
2. Install dependencies:
   ```
   pip install -r requirements.txt
   ```
3. Add environment variables: Create a `.env` file with your Groq API key:
   ```
   GROQ_API_KEY=your_api_key_here
   ```
4. Run the application:
   ```
   streamlit run app.py
   ```

Usage

1. Go to the "Analyze & Input" page
2. Upload or paste code
3. Click **Analyze Code** for results
4. Use **Format Code** (Python only)
5. Chat with the AI in the "Chat" section
6. View past reports in the "History" section

Dependencies

- Python 3.9 or higher
- streamlit >= 1.28.0
- langchain >= 0.0.350
- langchain-groq
- python-dotenv
- black