A SYNOPSIS OF MINOR PROJECT II ON

Clarity

Submitted In Partial
Fulfillment Of the Requirements for The Award of The Degree Of

BACHELOR OF TECHNOLOGY

Computer Science & Engineering-Artificial Intelligence

and Machine Learning

By

NIPUN LAKHERA (0002AL231047)
SAHIL RAIKWAR (0002AL231055)
MO ZAID SHEIKH (0002AL231042)
SHIVANSH NIGAM (0002AL231062)

GUIDE

Mr Vipin Verma

Co-GUIDE

Ms Swati Patel

School of Information Technology
Rajiv Gandhi Proudyogiki Vishwavidyalaya
Bhopal
July - December 2025

1. Introduction

The rapid evolution of software development demands not only functional code but also code that is robust, efficient, and easily maintainable. Developers frequently encounter challenges in debugging and optimizing code, leading to increased development time and potential errors in production. This project, Clarity, addresses these challenges by proposing an innovative AI-powered code refinement system designed to automatically identify and correct inefficiencies or bugs in Python code.

Clarity is being developed as a web-based application, leveraging a modern technology stack to deliver a seamless user experience. The backend is built using Python with the FastAPI framework, chosen for its high performance and robust API development capabilities. At its core, Clarity utilizes advanced Artificial Intelligence (AI) through Hugging Face Transformers, specifically employing a pre-trained language model such as Salesforce/codet5-small that is fine-tuned for code generation and understanding. The frontend will consist of HTML, CSS, and JavaScript, providing an intuitive interface where users can input their Python code and receive refined suggestions. This project operates at the intersection of Artificial Intelligence and Software Engineering, aiming to enhance developer productivity and code quality through intelligent automation.

2. Rationale

In the lifecycle of software development, writing functional code is only the first step. A significant amount of a developer's time is dedicated to refining this code—improving its readability, ensuring consistent naming conventions, and performing simple reviews to enhance clarity and maintainability. While working on projects, developers often seek a quick and accessible way to get a "second opinion" on their code snippets to ensure they adhere to best practices. This iterative process of refinement is crucial for long-term project health but can be time-consuming when performed manually.

While many powerful AI-driven coding assistants and large language models exist, they are often integrated into complex Integrated Development Environments (IDEs) or are designed for generating large blocks of new code from natural language prompts. There is a distinct need for a more focused, lightweight tool specifically designed for the task of code refinement. The motivation behind Clarity is to fill this gap, providing a simple, web-based utility where a developer can instantly submit a piece of code and receive targeted feedback on aspects like variable naming, logical structure, and bug correction. This approach offers an immediate and accessible solution for improving code quality without the overhead of more complex tools.

3. Objectives

The primary objectives of the "Clarity" project are as follows:

- To design and implement a robust backend API using Python and FastAPI capable of receiving, processing, and returning refined Python code snippets.
- To integrate a pre-trained AI model from Hugging Face Transformers (Salesforce/codet5-small) into the backend to intelligently identify and suggest corrections for syntax errors, logical bugs, and adherence to Python coding conventions.
- To develop an intuitive and user-friendly web-based frontend using HTML, CSS, and JavaScript, enabling developers to easily submit their code and visualize the AI-generated refinements.

- To evaluate the effectiveness and practical utility of the "Clarity" system in enhancing code quality and assisting developers in streamlining their code review and refinement processes.

4. Feasibility Study

The feasibility study for the "Clarity" project assesses its viability across technical, operational, and economic dimensions, affirming its potential to deliver a valuable solution within the project's scope and timeframe. This initial phase of software engineering ensures that the proposed solution is not only desirable but also achievable with available resources and technology.

Technical Feasibility: The project leverages well-established and mature technologies. The backend, built with Python and FastAPI, provides a robust and performant foundation for API development. The core AI functionality relies on the Hugging Face Transformers library, which offers access to pre-trained, state-of-the-art models like Salesforce/codet5-small that are specifically designed for code-related tasks. This eliminates the need for extensive model training from scratch, making the AI integration technically feasible for a minor project. The frontend will be developed using standard web technologies (HTML, CSS, JavaScript), which are highly accessible and require no complex build setups for the MVP. All chosen technologies are open-source and have extensive community support, mitigating potential technical hurdles.

Operational Feasibility: "Clarity" is designed to directly address a prevalent operational need among developers: the efficient refinement and review of Python code snippets. Its web-based interface will provide a user-friendly and accessible tool that integrates seamlessly into a developer's workflow, without requiring specialized software installations beyond a web browser. The system's ability to offer immediate feedback on code quality, naming conventions, and potential bugs directly contributes to improved operational efficiency and reduced manual review efforts.

Economic Feasibility: The project development relies entirely on free and open-source software and libraries, minimizing direct development costs. The use of pre-trained AI models and the lightweight nature of the frontend reduce computational resource requirements during both development and potential deployment phases. This approach ensures that the project can be completed within the typical resource constraints of a university minor project, making it economically viable.

Schedule Feasibility: The project roadmap, divided into distinct phases from backend skeleton to AI integration, frontend development, and eventual deployment, ensures a structured and manageable approach. The incremental nature of the development, coupled with the focused scope on Python code refinement, makes the project achievable within the academic semester's timeline (July - December 2025).

Need: The fundamental need for "Clarity" arises from the universal challenge developers face in consistently producing high-quality, readable, and error-free code. Manual code reviews, especially for minor refinements or adherence to specific coding standards, can be time-consuming and subjective. A tool that can automate this process for snippets provides an invaluable resource, accelerating learning for new developers and boosting productivity for experienced ones.

Significance: The significance of "Clarity" lies in its potential to democratize code quality improvement. By offering an accessible AI-driven tool, it empowers individual developers and small teams to elevate the standard of their Python codebase. It serves as an educational aid for understanding best practices and a practical utility for ensuring consistency and correctness. Furthermore, it demonstrates the tangible application of AI in enhancing core software engineering practices, bridging the gap between theoretical AI models and practical developer tools.

5. Methodology/ Planning of Work

The development of the "Clarity" project will follow an iterative and agile methodology, focusing on delivering a Minimum Viable Product (MVP) that incrementally builds functionality. This approach allows for continuous feedback, adaptation, and efficient resource utilization throughout the development cycle. The project's structure is modular, dividing the work into distinct units: the Backend (API and AI service) and the Frontend (User Interface).

Research Type & Approach: The project primarily involves applied research and development. It applies existing AI models and software development frameworks to solve a practical problem. The approach will be iterative, with each phase building upon the successful completion of the previous one, allowing for early integration testing and refinement.

Methods and Tools:

1. Version Control: Git and GitHub will be used for collaborative development, code management, and tracking changes across the team.
2. Backend Development:
    - Language & Framework: Python with FastAPI for building robust and high-performance RESTful APIs.
    - AI Integration: The transformers library from Hugging Face will be used to load and utilize a pre-trained text2text-generation model (e.g., Salesforce/codet5-small). This involves crafting appropriate prompts to guide the AI for effective code refinement and error correction.
    - Dependency Management: pip (or uv for faster installation) and requirements.txt will manage project dependencies.
3. Frontend Development:
    - Technologies: HTML for structure, CSS for styling, and JavaScript for interactive logic, ensuring a lightweight and universally compatible user interface.
    - API Communication: The Fetch API will be used in JavaScript to asynchronously communicate with the FastAPI backend.
4. Testing & Quality Assurance:
    - Unit Testing: Python's unittest or pytest framework may be employed for unit testing critical backend components.
    - Integration Testing: Thorough testing will be conducted to ensure seamless communication and data flow between the frontend and backend, as well as the successful integration of the AI model.
    - User Acceptance Testing (UAT): Early prototypes will be tested by potential users to gather feedback on usability and effectiveness, informing subsequent iterations.

Planning of Work (Project Phases): The development will be structured according to the following phases, as outlined in the project's detailed roadmap:

- Phase 1: The Backend Skeleton: Establishing the foundational FastAPI application (main.py), implementing CORS middleware, defining the API endpoint (/api/correct), and creating a mock AI service (model_service.py) to return predictable responses. This phase focuses on setting up the core communication infrastructure.

- Phase 2: The Frontend User Interface: Developing the client-side application (index.html, script.js, style.css) with input/output areas and a submission button. This phase focuses on enabling user interaction and connecting to the mock backend
- Phase 3: The AI Brain: Replacing the mock AI service with the actual Hugging Face Transformer model in model_service.py. This includes loading the model, crafting AI prompts, and processing its output to refine Python code. This is the core intelligence integration phase.
- Phase 4: Integration and Polish: Connecting the fully functional frontend with the AI-powered backend. This phase involves implementing loading states, robust error handling, and refining the user interface and experience for a professional appearance.
- Phase 5: Deployment: Deploying the FastAPI backend to a platform like Hugging Face Spaces and the static frontend to services like Vercel or GitHub Pages, making the application publicly accessible.

This systematic approach ensures that each component of "Clarity" is developed, tested, and integrated effectively, leading to a functional and impactful final product.

6. Software/Hardware Required for the Project's Development

The successful development and deployment of the "Clarity" project necessitate a combination of modern software tools and standard hardware specifications. The choice of technologies emphasizes open-source solutions to ensure accessibility and minimize development costs.

Software Requirements:

1. Operating System:
   - Development: Linux (e.g., Ubuntu, Debian), macOS, or Windows (via WSL2 for Linux environment compatibility).
   - Deployment: Linux-based environments (e.g., Docker containers on cloud platforms).
2. Programming Languages:
   - Python (3.8+): For backend logic, API development, and AI model integration.
   - JavaScript: For interactive frontend development.
   - HTML5 & CSS3: For structuring and styling the web-based user interface.
3. Backend Frameworks & Libraries:
   - FastAPI: A modern, fast (high-performance) web framework for building the API.
   - Uvicorn: An ASGI server to run the FastAPI application.
   - Hugging Face Transformers: Library for integrating pre-trained AI models (Salesforce/codet5-small).
   - PyTorch (or TensorFlow): Underlying deep learning framework required by Hugging Face Transformers.
4. Frontend Technologies:
   - Web Browser (Chrome, Firefox, Edge, Safari): For testing and interacting with the web application.
5. Development Tools:
   - Git: Distributed version control system for collaborative development.
   - GitHub: For hosting the project's codebase and facilitating team collaboration.
   - Code Editor/IDE: Visual Studio Code, PyCharm, or similar, for writing and debugging code.

- Python Virtual Environment: venv (or uv) for managing project-specific Python dependencies.
- pip (or uv): Python package installer.

6. Deployment Platforms:
   - Hugging Face Spaces: For hosting the AI-powered FastAPI backend.
   - Vercel / GitHub Pages: For hosting the static frontend application.

Hardware Requirements:

1. Development Machine:
   - Processor: Multi-core CPU (Intel i5/Ryzen 5 equivalent or better).
   - RAM: 8 GB (16 GB recommended for smoother AI model loading and execution).
   - Storage: 256 GB SSD (512 GB recommended), with sufficient free space for IDEs, project files, and AI model downloads.
   - Internet Connection: Stable broadband connection for downloading dependencies and AI models, and for accessing online resources.
2. AI Model Inference (Beneficial but not strictly required for \`codet5-small\`):
   - GPU: A dedicated GPU (e.g., NVIDIA with CUDA support) is highly beneficial for accelerating AI model inference, especially if larger models are considered or if the application experiences high traffic, but the Salesforce/codet5-small model can run effectively on modern CPUs for prototype purposes.

7. Expected Outcomes

Upon the successful completion of the "Clarity" project, the following key outcomes are anticipated:

- A Functional Web-Based AI Code Refiner: The primary outcome will be a fully operational web application accessible via a browser, allowing users to submit Python code snippets for AI-driven analysis and refinement.
- Automated Code Quality Improvement: The system will effectively identify and suggest corrections for common programming errors, enhance adherence to best practices (e.g., naming conventions), and improve code efficiency and readability.
- Enhanced Developer Productivity: By providing instant, intelligent code review and refinement, "Clarity" is expected to significantly reduce the time developers spend on debugging and manual code improvements, thereby boosting overall productivity.
- Practical Application of AI in Software Engineering: The project will serve as a tangible demonstration of how advanced AI models can be effectively integrated into developer tools to automate complex tasks and provide intelligent assistance.
- * Educational Value: For the development team, the project provides invaluable hands-on experience with modern full-stack development, AI model integration, and cloud deployment strategies. For users, it offers an interactive platform for learning and applying good coding practices.